Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Pečnik

Avtomatiziran sistem za priporočanje receptov iz spletišč

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Aleš Smrdel

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Pogosto se zgodi, da imamo v hladilniku ali omari izdelke in sestavine, iz katerih bi radi naredili neko jed, pa ne vemo, katero jed bi naredili, ali pa ne poznamo točnega recepta za to jed. Da bi se temu izognili, razvijte priporočilni sistem, ki bo poiskal in priporočil recepte za jedi, ki jih je mogoče narediti na podlagi izbranih izdelkov oziroma sestavin. Sistem naj omogoča vnašanje izdelkov na podlagi imena izdelka na računu ali prebrane črtne kode izdelka. V ta namen izdelajte tudi mobilno aplikacijo, ki bo omogočala zajem slike in razpoznavanje izdelkov na podlagi optičnega prepoznavanja znakov oziroma branje črtne kode z zajete slike. Sistem naj na podlagi vnesenih izdelkov prepozna sestavine in poišče in predlaga recepte za jedi, ki jih je mogoče narediti z uporabo izbranih sestavin. Pri izdelavi sistema izberite najprimernejše tehnologije tako na strani strežnika in odjemalca kot tudi za izdelavo mobilne aplikacije.

Izjava o avtorstvu zaključnega dela

Spodaj podpisani Matej Pečnik, vpisna številka **63120105**, avtor zaključnega dela z naslovom:

Avtomatiziran sistem za priporočanje receptov iz spletišč
(angl. *Automated system to recommend recipes from websites*)

IZJAVLJAM

1. da sem pisno zaključno delo študija izdelal samostojno pod mentorstvom doc. dr. Aleša Smrdela;
2. da je tiskana oblika pisnega zaključnega dela študija istovetna elektronski obliki pisnega zaključnega dela študija;
3. da sem pridobil/-a vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v pisnem zaključnem delu študija in jih v pisnem zaključnem delu študija jasno označil/-a;
4. da sem pri pripravi pisnega zaključnega dela študija ravnal/-a v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil/-a soglasje etične komisije;
5. soglašam, da se elektronska oblika pisnega zaključnega dela študija uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;
6. da na UL neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve avtorskega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja pisnega zaključnega dela študija na voljo javnosti na svetovnem spletu preko Repozitorija UL;
7. dovoljujem objavo svojih osebnih podatkov, ki so navedeni v pisnem zaključnem delu študija in tej izjavi, skupaj z objavo pisnega zaključnega dela študija.

V Ljubljani, dne 23. avgusta 2016 Podpis študenta/-ke:

Zahvaljujem se mentorju doc. dr. Alešu Smrdelu za vso pomoč pri delu ter hitre odzive in jasne odgovore. Posebej pa se zahvaljujem svoji punci Ani ter svojima staršema, mami Emiri in očetu Jožetu, za vso pomoč in podporo tekom študija.

Diplomsko delo posvečam pokojnemu dedku, Ata Ivanu.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
AJAX	Asynchronous JavaScript and XML	Asinhroni JavaScript in XML
API	Application Programing Interface	Aplikacijski programski vmesnik
CRUD	Create, read, update and delete	Ustvari, beri, posodobi, izbriši
CSS	Cascading Style Sheets	Kaskadne slogovne predloge
D3	Data-Driven Documents	Podatkovno usmerjeni dokumenti
FTP	File Transfer Protocol	Protokol za prenos datotek
HTML	Hyper Text Markup Language	Hipertekstovni označevalni jezik
HTTP	HyperText Transfer Protocol	Hipertekstovni prenosni protokol
IDE	Integrated Development Environment	Integrirano razvojno okolje
JSON	JavaScript Object Notation	Notacija objektov JavaScript
OCR	Optical Character Recognition	Optično prepoznavanje znakov
REST	Representational State Transfer	Prenos predstavitvenega stanja
SQL	Structured Query Language	Strukturirani povpraševalni jezik

URI	Uniform Resource Identifier	Enolični identifikator vira
URL	Uniform Resource Locator	Enolični krajevnik vira
XML	Extensible Markup Language	Razširljivi označevalni jezik

Povzetek

Naslov: Avtomatiziran sistem za priporočanje receptov iz spletišč

V diplomskem delu predstavimo razvoj in delovanje avtomatiziranega sistema, katerega glavna naloga je priporočanje receptov na podlagi izdelkov in sestavin, ki jih imamo v shrambi. V ta namen smo razvili komponente strežniškega dela ter spletno in mobilno aplikacijo. Ena izmed komponent je iskalnik sestavin, ki s pomočjo drevesa sestavin za določen vhod ugotovi, za katero sestavino gre. Za pridobitev podatkov o izdelkih in receptih sistem uporablja luščilnik, ki iz nekega spletnega vira pridobi podatke ter jih strukturirano shrani v podatkovno zbirko. Poleg luščilnika vsebuje sistem tudi iskalnik receptov in iskalnik izdelkov na računu. Prvi skrbi za izpis vseh receptov, ki jih lahko naredimo iz sestavin in izdelkov, ki jih imamo. Drugi pa poišče vse izdelke, ki smo jih pridobili s pomočjo optičnega prepoznavanja znakov skeniranega računa. Razvili smo tudi aplikacijski programski vmesnik, preko katerega strežniški del komunicira z odjemalci. Spletna aplikacija omogoča različne funkcionalnosti kot so: pregled izdelkov in sestavin shrambe, pregled receptov, pregled priljubljenih receptov. Poleg tega omogoča tudi dodajanja izdelkov in sestavin na različne načine. Mobilna aplikacija poleg vseh funkcionalnosti, ki jih omogoča spletna aplikacija, omogoča še dodajanje izdelkov po črtni kodi in po računu. Obe funkcionalnosti uporabljata vgrajeno kamero mobilne naprave.

Ključne besede: spletna aplikacija, mobilna aplikacija, OCR, priporočanje receptov, luščilnik podatkov, API, drevo sestavin.

Abstract

Title: Automated system to recommend recipes from websites

In our diploma thesis we present the development and functioning of automated system, which main task is to recommend the recipes on the basis of the products and ingredients, which we have in our pantry. For this purpose we developed the components of the server part as well as web and mobile application. One of the components is the search engine of the ingredients, which, by means of the tree of ingredients, for a specific input determines which is the ingredient. In order to obtain the data the system uses the scraper which acquires data from a web source and saves them in the database in a structured way. In addition to the scraper, the system also contains the search engine of the recipes and the search engine of the products within the bill. The first one is responsible for the display of all the recipes which can be made of the ingredients and products which we have. The other one determines all the products which we acquired by means of optical character recognition of the characters of the scanned bill. We also developed application programming interface through which the server part communicates with the clients. Web application enables different functionalities, such as: review of the products and ingredients of the pantry, review of the recipes, review of the favorite recipes. In addition, also enables adding the products and ingredients in different ways. Mobile application in addtion to all the functionalities enabled by web application, it also enables adding products by means of a bar code and by means of a bill. Both functionalities use the built-in camera of the mobile device.

Keywords: web application, mobile application, OCR, recommending recipes, data scraper, API, tree of ingredients.

Poglavje 1

Uvod

Vsi imamo doma določene sestavine, iz katerih lahko pripravimo razne jedi. Vendar pogosto nimamo idej, oziroma smo omejeni na jedi, ki jih že poznamo in jih pogosto pripravljamo. Hkrati se znajdemo tudi pod časovnim pritiskom, ko bi morali jed pripraviti zelo hitro. Že sama priprava jedi zahteva določen čas, poleg tega pa lahko tudi iskanje ustreznega recepta za pripravo vzame preveč časa. Obenem bi morda radi še porabili katero izmed sestavin, katerim se bliža iztek roka uporabnosti. Vendar pa ne vemo, kako bi jo vključili v našo jed, oziroma katero jed, ki vsebuje izbrane sestavine, bi pripravili. V diplomskem delu predstavljamo celoten sistem, ki poskuša rešiti zgoraj navedene probleme. Sistem je sestavljen iz priporočilnega sistema, ki teče na strežniku, spletne aplikacije, ki služi kot odjemalec, ter tudi mobilne aplikacije, ki služi kot odjemalec na mobilnih napravah.

1.1 Pregled sorodnih rešitev

Na spletu najdemo kar nekaj spletnih aplikacij, ki nam predlagajo recepte na podlagi prej vnesenih sestavin. Vsaka od aplikacij, ki jih bomo v nadaljevanju predstavili, vsebuje tako pozitivne, kot tudi negativne lastnosti.

Spletna aplikacija **mizicapogrnise.si** [22] je nastala kot rezultat diplomskega dela z naslovom *Podobnostne mreže receptov in spletna aplikacija za priporočanje sestavin jedi* [14] na Fakulteti za računalništvo in informatiko v Ljubljani. Aplikacija deluje tako, da po vpisu sestavin v iskalnik uporabniku ponudi recepte, na podlagi katerih lahko naredimo jedi iz vnesenih sestavin. Uporabnik ima poleg vnosa sestavin možnost izbire tipa jedi in razvrščanje receptov glede na pomembnost sestavine ter glede na število sestavin, ki jih ima. Pomanjkljivost aplikacije je omejitev nabora receptov. Pri iskanju primernih receptov aplikacija uporablja samo recepte, ki so dosegljivi na spletni strani *mojirecepti.com* [23]. Hkrati je slabost aplikacije tudi, da pri ponovnem obisku strani ne ohrani prej vnesenih sestavin.

Spletni portal **okusno.je** [27] je namenjen obiskovalcem, ki iščejo različne kuharske recepte, ideje, namige in nasvete. Aplikacija "Hladilnik" ponudi obiskovalcem določene recepte na podlagi vnesenih sestavin. Pomanjkljivost aplikacije je, podobno kot pri *mizicapogrnise.si*, omejen nabor receptov, saj uporablja lokalno zbirko receptov. Poleg tega ima aplikacija omejitev vnosa do največ pet sestavin. Slabost je tudi vnos sestavin preko prej generiranega spustnega seznama, kar povzroči dolgotrajno iskanje ter vnašanje sestavin. Prav tako kot pri prejšnji aplikaciji, se pri ponovnem obisku portala ne ohranijo prej vnesene sestavine.

Naslednja spletna aplikacija, **jedel.bi** [15], je zasnovana kot iskalnik okusnih jedi. Uporabnik s pomočjo iskalnika vnese sestavine, ki jih aplikacija nato uporabi pri predlaganju jedi. Aplikacijo je možno uporabljati tako v spletnem brskalniku kot tudi na mobilnih napravah, na katerih teče opera-

cijski sistem iOS ali Android. Tako kot večina prej opisanih aplikacij ima tudi ta aplikacija že omenjeni pomanjkljivosti, saj uporablja lokalno zbirko podatkov ter pri ponovnem obisku portala ne ohrani prej vnesenih sestavin. Poleg naštetega je aplikacija tudi precej nepregledna.

Naslednji spletni portal **kulinarika.net** [16] je nastal v času, ko kuharske spletne strani še niso bile tako razširjene, in sicer leta 1998. *Kulinarika.net* je danes največji kulinarični spletni portal v Sloveniji z več kot pol milijona različnimi obiskovalci mesečno. *Kulinarika.net* ima poleg iskanja receptov na podlagi imena tudi možnost naprednega iskanja, ki omogoča iskanje receptov na podlagi vnesenih sestavin. Rezultat iskanja so vsi recepti, kjer se določena vnesena sestavina pojavi. Tako kot večina že prej omenjenih aplikacij ima tudi *kulinarika.net* to pomanjkljivost, da uporablja le lokalno zbirko receptov, ter da se po ponovnem obisku portala ne ohranijo prej vnesene sestavine.

1.2 Cilj diplomskega dela

Cilj diplomskega dela je razviti avtomatiziran sistem, ki bo na podlagi vnesenih sestavin ali izdelkov kupljenih v trgovini, priporočal recepte, ki jih lahko uporabnik naredi. Avtomatiziran sistem bo sestavljen iz več komponent. Ena izmed komponent bo iskalnik sestavin. Naloga te komponente bo, da bo s pomočjo algoritma ter drevesa sestavin, na podlagi imena izdelka kupljenega v trgovini, ugotovila za katero sestavino gre pri določenem izdelku (npr. izdelek: svinjski kare zrezek, Anton, 500g, pakirano → sestavina: meso - svinjina - kare). Hkrati bo sistem vseboval komponento spletni luščilnik podatkov, katere naloga bo, da bo iz različnih spletnih virov pridobila podatke o izdelkih in receptih ter jih strukturirano shranila v podatkovno zbirko. Komponenta iskalnik receptov bo skrbela, da bo na podlagi sestavin, ki jih imamo v shrambi, poiskala vse recepte, ki jih lahko iz teh sestavin naredimo. Komponenta iskalnik izdelkov na računu pa bo za podan vhod, ki bo predstavljal niz izdelka iz računa, pridobljen s pomočjo OCR (ang. Optical Character Re-

cognition), poiskala izdelek, ki pripada temu nizu. Poleg avtomatiziranega sistema je cilj diplomske naloge razviti spletno in mobilno aplikacijo, preko katerih bomo dobili celoten pregled nad trenutno vnesenimi sestavinami ter recepti, ki jih lahko naredimo. Spletna aplikacija bo nudila vnos sestavin po imenu izdelka, sestavine ali pa črtne kode izdelka, ki je prisotna na vsakem izdelku, kupljenem v trgovini. Mobilna aplikacija pa bo poleg vseh funkcionalnosti, ki jih zagotavlja spletna aplikacija, omogočala vnos izdelkov preko branja črtnih kod ter skeniranja računa. Pri obeh načinih vnosa se bo uporabljala vgrajena kamera mobilne naprave. Skeniranje računa bo vsebovalo še optično prepoznavanje znakov (OCR). Za komunikacijo med odjemalci ter podatkovno zbirko pa bo poskrbel aplikacijski programski vmesnik (API), ki bo s pomočjo funkcionalnosti strežniškega dela odjemalcem odgovarjal na željene zahteve.

1.3 Struktura dela

Diplomsko delo je razdeljeno v šest poglavij. Uvodnemu delu sledi Poglavje 2, kjer opišemo in predstavimo tehnologije in orodja, ki smo jih uporabili za realizacijo celotne rešitve, sestavljene iz strežniškega dela, spletne aplikacije in mobilne aplikacije. V Poglavju 3 predstavimo idejo ter realizacijo podatkovne strukture za hranjenje sestavin ter podatkovno zbirko, ki je sestavljena iz treh skupin. Sledi Poglavje 4, v katerem predstavimo delovanje vsake od komponent avtomatiziranega sistema. Poleg komponent avtomatiziranega sistema v tem poglavju predstavimo še delovanje APIja, spletne aplikacije ter mobilne aplikacije. V Poglavju 5 predstavimo uporabo spletne in mobilne aplikacije. V zadnjem poglavju, Poglavju 6, pa povzamemo rezultate ter podamo ideje za nadaljnje delo.

Poglavje 2

Uporabljene tehnologije in orodja

V tem poglavju predstavljamo tehnologije in orodja, ki smo jih uporabili za razvoj strežniškega dela, spletne aplikacije in mobilne aplikacije.

2.1 Strežniški del

Na strani strežnika smo uporabili naslednje tehnologije:

PHP (angl. PHP: Hypertext Preprocessor) je skriptni jezik, ki je namenjen za obdelavo podatkov ter gradnjo dinamičnih in interaktivnih spletnih strani [10]. Programi oziroma skripte imajo končnico *.php* ter se s pomočjo PHP interpreterja vsakič v celoti procesirajo na spletnem strežniku. Rezultat procesiranja je ponavadi dokument HTML ali del dokumenta HTML, ki se prenese na odjemalca in se tam tudi prikaže.

MySQL je ena najpopularnejših relacijskih podatkovnih zbirk, ki deluje po principu odjemalec/strežnik. Vključuje strežnik SQL, programe za dostop do strežnika, administrativna orodja ter programski vmesnik za pisanje lastnih programov [11].

MySQL Workbench je vizualno orodje za gradnjo arhitekture podatkovnih zbirk ter razvoj in upravljanje s podatkovnimi zbirkami [24].

Python je skriptni jezik, ki omogoča hitro programiranje tako enostavnih kot tudi kompleksnih programov, saj ima poleg enostavne sintakse tudi visokonivojske podatkovne strukture, ki so tesno vdelane v sam jezik. Prav tako pa Python vsebuje še elemente funkcijskega programiranja ter module, ki programiranje pogosto spremenijo v komaj kaj več kot lepljenje tuje kode [9].

Requests je knjižnica, ki je namenjena enostavni interakciji s spletom v programskem jeziku Python. Pri svojem delovanju uporablja knjižnico urllib3, ki je vdelana v zahtevkih. Omogoča nam enostavno pošiljanje zahtevkov HTTP brez potrebe po ročnem dodajanju poizvedbenih nizov ali kodiranju poslanih podatkov. Poleg tega vsebuje tudi funkcionalnosti kot so: ohranjanje povezav (angl. keep-alive), vztrajnost sej in piškotkov, časovne omejitve (angl. Connection Timeouts) in druge [7].

Beautiful Soup je knjižnica, ki nam s pomočjo programskega jezika Python omogoča pridobivanje podatkov iz dokumentov HTML ter XML. Vsak dokument HTML ter XML lahko s pomočjo te knjižnice pretvorimo v objekt, ki temelji na različnih lastnostih ter metodah. Preko teh lastnosti in metod lahko enostavno izluščimo iskane podatke [25].

PyMySQL je vmesnik za dostop do podatkovne zbirke MySQL preko programskega jezika Python. Implementira Python programski vmesnik ter vsebuje knjižnico za dostop do podatkovne zbirke MySQL [28].

2.2 Spletna aplikacija

Na strani spletne aplikacije smo uporabili naslednje tehnologije:

HTML (angl. Hypertext Markup Language) je označevalni jezik, ki opisuje, kako so besedila, grafike in datoteke, ki vsebujejo druge podatke, organizirani in povezani [19]. Dokument HTML je sestavljen iz posebnih značk, ki brskalniku povejo, kako naj določen del vsebine prikaže.

CSS (angl. Cascading Style Sheets) je slogovni jezik, ki definira slog konstruktov kot so na primer pisava, barva in pozicioniranje, ki se uporabljajo za opis, kako je informacija na spletni strani oblikovana in prikazana [19]. Slog CSS je možno v dokument HTML vključiti direktno ali pa preko posebne slogovne datoteke, ki ima končnico *.css*.

Bootstrap je eden najbolj uporabljenih ogrodij za gradnjo spletnih strani na svetu. Uporablja nabor najnovejših spletnih tehnologij (HTML5, CSS3, knjižnico jQuery) in s tem zagotavlja zanesljivo delovanje vgrajenih funkcionalnosti, kot so mrežno razvrščanje, prilagodljiv slog na vseh napravah, tipografijo ter še mnogo drugih [13].

JavaScript je skriptni jezik, ki se najpogosteje uporablja v kombinaciji z označevalnim jezikom HTML. Omogoča dinamično ter interaktivno spreminjanje ali urejanje vsebine na spletni strani [12]. JavaScript kodo lahko vgradimo neposredno v dokument HTML, ali pa kodo v dokument HTML vključimo preko datotek s skriptno kodo. Pri sami vključitvi kode v dokument HTML moramo JavaScript kodo hraniti v datoteki s končnico *.js*.

jQuery je knjižnica, ki je napisana v programskem jeziku JavaScript. Ponuja nam preprost način za hitro in dosledno doseganje istih ciljev kot z JavaScriptom, vendar preglednejše in z manj vrsticami kode [12]. Moto knjižnice jQuery je “Napiši manj, naredi več”.

D3 (angl. Data-Driven Documents) je JavaScript knjižnica namenjena vizualizaciji podatkov. Omogoča predstavitev podatkov v obliki grafov, dreves ter drugih struktur brez potrebnih skrbi o izračunih, položajih ter velikostih elementov, ki so sestavni deli grafov oziroma struktur [32].

2.3 Mobilna aplikacija

Na strani mobilne aplikacije smo uporabili naslednje tehnologije:

Xcode je integrirano razvojno okolje (angl. Integrated Development Environment), ki se uporablja za razvoj in testiranje aplikacij, ki tečejo na Applovih operacijskih sistemih OS X, iOS, watchOS in tvOS. Okolje vsebuje vsa orodja ter funkcionalnosti za celovit razvoj in testiranje aplikacije. Sestavljajo ga naslednje komponente: urejevalnik, pregledovalnik, generator vmesnikov za aplikacijo, razhroščevalnik, upravljalec z različicami ter orodje za testiranje enot [26].

Swift je programski jezik, ki je namenjen izdelavi aplikacij, ki se izvajajo na Applovih operacijskih sistemih iOS, OS X, watchOS in tvOS. Temelji na programskih jezikih C in Objective-C, kar mu omogoča popolno združljivost z obema programskima jezikoma brez omejitev. Zasnovan je za delo z Applovim ogrodjem Cocoa in Cocoa Touch, ki skrbita za gradnjo uporabniških vmesnikov aplikacij [1].

Tesseract je odprtokoden sistem za optično prepoznavanje znakov. Razvilo ga je ameriško podjetje HP (Hewlett-Packard) med leti 1984 in 1994. Tesseract se je začel razvijati kot doktorski raziskovalni projekt v laboratoriju podjetja HP, katerega namen je bil strojni dodatek oziroma program v optičnih bralnikih HP. Vse do leta 2005 Tesseract ni doživel resnejših posodobitev. Potem pa se je leta 2005 podjetje HP odločilo, da sprosti njegovo

izvorno kodo. Danes Tesseract deluje pod okriljem spletnega velikana Google, ki je prav tako javno objavil izvorno kodo tega projekta [29]. Tesseract je napisan v programskih jezikih C in C++. Poleg tega pa najdemo na spletu tudi Tesseract ogrodje za operacijski sistem iOS, ki omogoča uporabo Tesseract funkcij v programskih jezikih Objective-C in Swift.

AVFoundation je ogrodje, ki omogoča delo z avdio ter video mediji v aplikacijah, ki se izvajajo na operacijskih sistemih iOS in Mac OS X. AVFoundation zagotavlja tako Objective-C kot tudi Swift vmesnik, preko katerega lahko enostavno uporabljamo funkcije, kot so igranje, snemanje, urejanje ali kodiranje medijskih formatov [2].

Poglavje 3

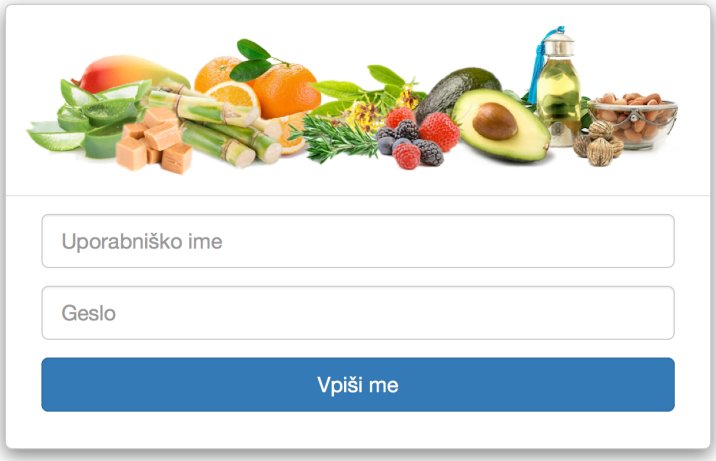
Podatkovni model

V tem poglavju bomo predstavili glavne komponente za hranjenje podatkov. Najprej bomo predstavili drevo sestavin, ki predstavlja podatkovno strukturo za hranjenje sestavin. Nadaljevali bomo s predstavitvijo podatkovne zbirke, ki je sestavljena iz treh skupin. Podrobno bomo opisali vsako od skupin ter predstavili njeno vlogo. Na koncu tega poglavja bomo predstavili še tabelo Ingredient, ki predstavlja srce podatkovne zbirke, saj služi kot vozlišče predstavljenih skupin, v kateri je shranjeno drevo sestavin.

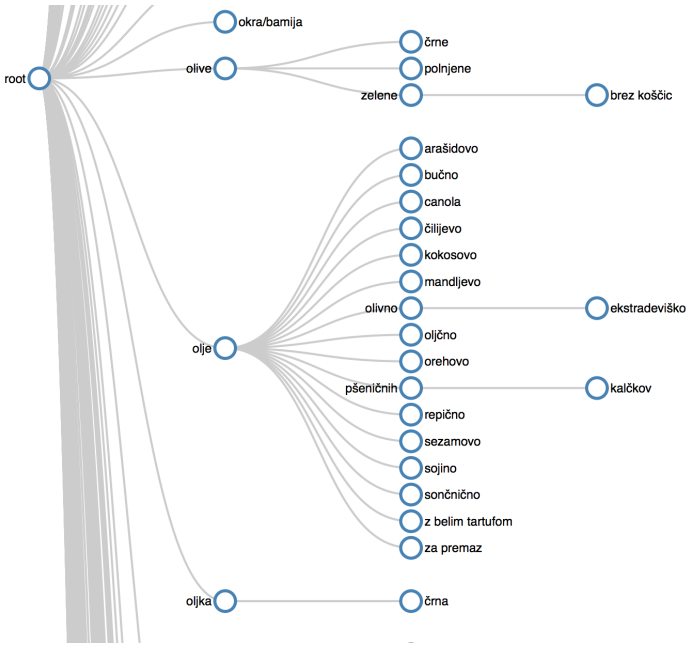
3.1 Drevo sestavin

Drevo sestavin predstavlja podatkovno strukturo za hranjenje sestavin. Gradnja drevesa je potekala tako, da smo programsko iz različnih spletnih virov zbrali približno 1200 sestavin ter jih shranili v strukturo JSON. Sledila je ročna delitev sestavljenih sestavin v nivoje. Poleg tega smo razvili tudi spletni uporabniški vmesnik za lažji pregled. Za grafični prikaz drevesa JSON smo uporabili JavaScript knjižnico *D3*, ki smo jo predstavili v podpoglavju 2.2. S pomočjo razvitega uporabniškega vmesnika za grafični prikaz drevesa sestavin so nam le-tega s svojim poznavanjem sestavin pomagali dopolnjevati tudi znanci in prijatelji. Na Sliki 3.1 je prikazano okno za prijavo v spletni vmesnik za prikaz drevesa sestavin, medtem ko je na Sliki 3.2 prikazan del

grafičnega prikaza drevesa sestavin. Vozlišče *root* predstavlja koren drevesa sestavin.



Slika 3.1: Vpisno okno v stran za grafični prikaz drevesa sestavin.



Slika 3.2: Grafični prikaz drevesa sestavin.

Posamezna sestavina je v strukturi JSON predstavljena z naslednjimi lastnostmi:

- **id**
  Predstavlja identifikator sestavine, ki je 15-mestno število, sestavljeno iz petih delov. Omogoča nam predstavitev drevesne strukture v podatkovni zbirki ter učinkovito iskanje po sami strukturi. Vsak izmed delov je 3-mestno število, ki predstavlja zaporedno številko sestavine oziroma podvrsto sestavine na tem nivoju.

$$\underbrace{054}_{1.\,nivo}\ \underbrace{001}_{2.\,nivo}\ \underbrace{002}_{3.\,nivo}\ \underbrace{001}_{4.\,nivo}\ \underbrace{001}_{5.\,nivo}$$

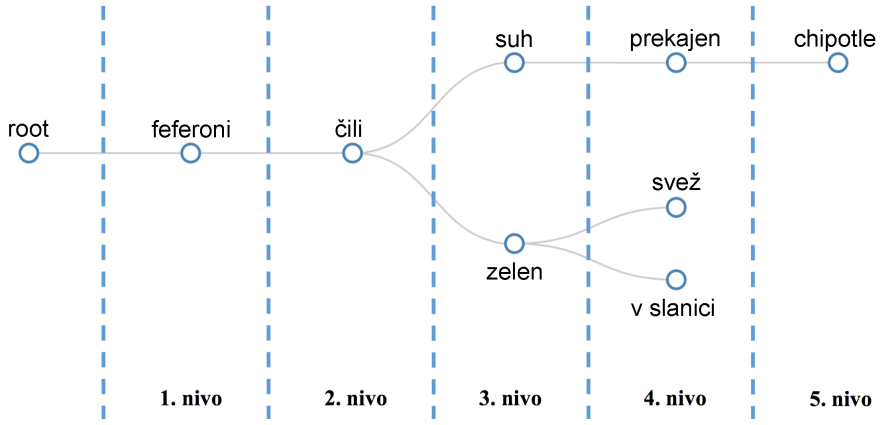
- **children**
  Predstavlja seznam otrok sestavine.
- **name**
  Predstavlja ime sestavine.
- **parent**
  Predstavlja ime starša.
- **depth**
  Predstavlja nivo na katerem se sestavina nahaja.

Koda 3.1 prikazuje predstavitev zapisa osnovne sestavine *kvas*.

Koda 3.1: Primer predstavitve sestavine v strukturi JSON.

```
{
    "id": "136000000000000",
    "children": [],
    "name": "kvas",
    "parent": null,
    "depth": 0
}
```

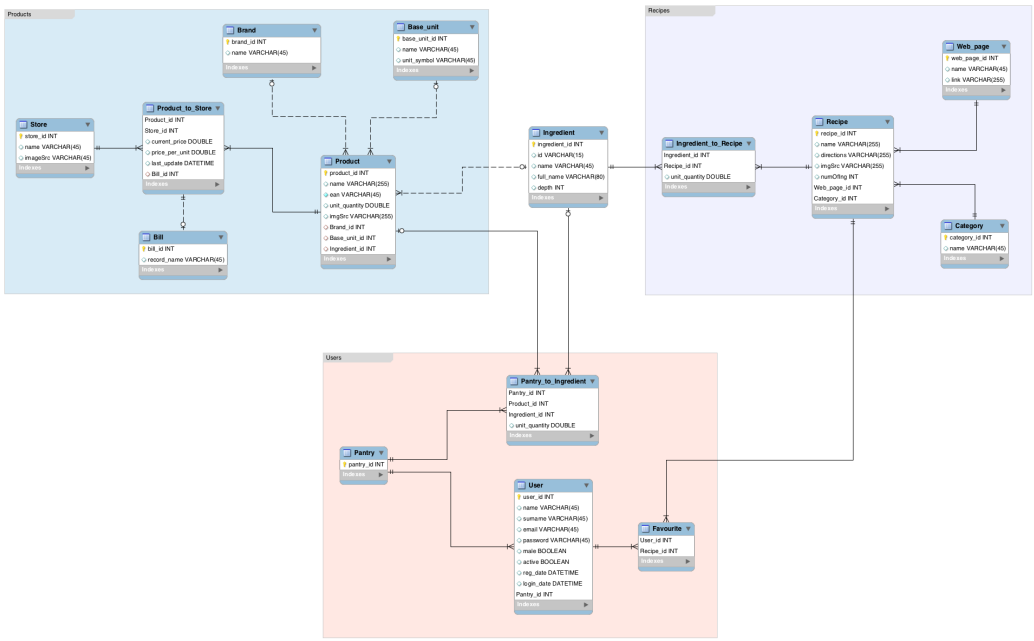
Na Sliki 3.3 lahko vidimo prikaz nivojev na grafičnem vmesniku. *Feferoni* so v prikazanem drevesu na prvem nivoju. Na drugem nivoju se nahaja ena izmed vrst feferona *čili*, ki pa se na naslednjem deli na *suh* in pa na *zelen*. Suh čili se naprej deli na *prekajen*, ki se deli na *chipotle*. Zelen pa se na naslednjem nivoju ponovno deli, in sicer na *svež* in tisti *v slanici*. Drevo sestavin je v podatkovni zbirki shranjeno v tabeli Ingredient, ki je predstavljena v podpoglavju 3.2.



Slika 3.3: Prikaz nivojev na grafičnem vmesniku.

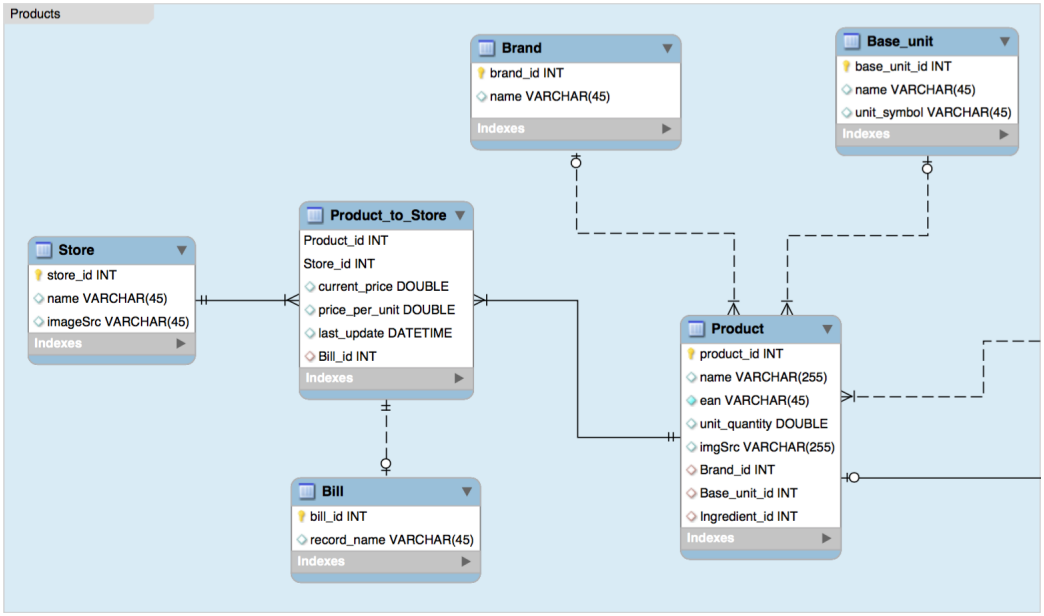
3.2 Podatkovna zbirka

Podatkovna zbirka predstavlja glavno zbirko podatkov, v kateri hranimo podatke o izdelkih, sestavinah, receptih ter uporabnikih. Pri realizaciji podatkovne zbirke smo uporabili relacijsko podatkovno zbirko MySQL, ki za delo s podatki uporablja povpraševalni jezik SQL (angl. Structured Query Language). V nadaljevanju bomo podrobneje predstavili funkcijo vsake od skupin. Slika 3.4 predstavlja podatkovni model, ki smo ga zgradili za potrebe našega sistema.



Slika 3.4: Prikaz podatkovnega modela.

Podatke o **izdelkih** kupljenih v trgovinah hranimo v tabelah skupine *Products*, ki jo prikazuje Slika 3.5. Skupina *Products* se nahaja na zgornjem levem delu Slike 3.4.



Slika 3.5: Prikaz tabel in relacij skupine *Products*.

Skupino sestavljajo naslednje tabele:

- **Product**
  Tabela Product predstavlja osrednjo tabelo skupine *Products*. V atributih:
  - *name* (predstavlja ime izdelka),
  - *ean* (predstavlja črtno kodo izdelka),
  - *unit_quantity* (predstavlja količino izdelka),
  - *imgSrc* (predstavlja povezavo na sliko izdelka)

  hrani osnovne podatke o izdelku.

- **Store**
  Tabela Store je namenjena hranjenju imena in logotipa trgovine, v kateri je bil neki izdelek kupljen. Te podatke hrani v atributih:
  - *name* (predstavlja ime trgovine),
  - *imageSrc* (predstavlja povezavo na logotip trgovine).

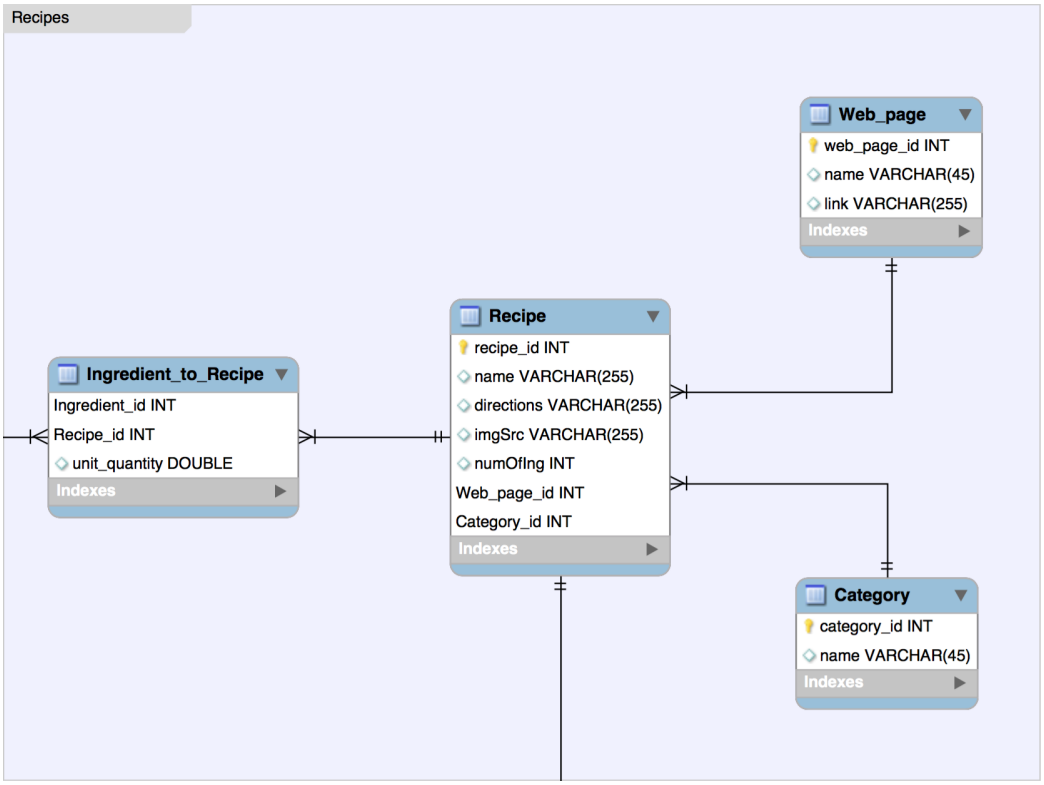
- **Product_to_Store**
  Tabela Product_to_Store predstavlja vmesno tabelo med tabelama Product in Store. Namenjena je hranjenju podatkov o ceni izdelka ter zadnji posodobitvi cene izdelka, kupljenega v neki trgovini. Tabela hrani podatke v atributih:
  - *current_price* (predstavlja ceno izdelka),
  - *price_per_unit* (predstavlja ceno izdelka na osnovno mersko enoto),
  - *last_update* (predstavlja datum in uro zadnje osvežitve cene izdelka).

- **Bill**
  Tabela Bill je namenjena hranjenju oznake izdelka na računu. Oznaka predstavlja zapis imena izdelka na računu, ki je bil izdan v neki trgovini. Oznako hrani v atributu *record_name*.

- **Brand**
  Tabela Brand hrani podatek o blagovni znamki izdelka. Vsebuje atribut *name*, ki predstavlja ime blagovne znamke.

- **Base_unit**
  Tabela Base_unit je namenjena hranjenju osnovnih merskih enot. Ta podatek hrani v atributih:
  - *name* (predstavlja polno ime merske enote) ter
  - *unit_symbol* (predstavlja kratico merske enote).

Podatke o **receptih** hranimo v tabelah skupine *Recipes*, ki jo prikazuje Slika 3.6. Skupina *Recipes* se nahaja na zgornjem desnem delu Slike 3.4.



Slika 3.6: Prikaz tabel in relacij skupine *Recipes*.

Skupino sestavljajo naslednje tabele:

- **Recipe**

  Tabela Recipe predstavlja osrednjo tabelo skupine *Recipes*. V atributih:

  - *name* (predstavlja ime recepta),
  - *directions* (predstavlja povezavo na spletni vir recepta),
  - *imgSrc* (predstavlja povezavo na sliko recepta),
  - *numOfIng* (predstavlja število sestavin, ki jih recept vsebuje)

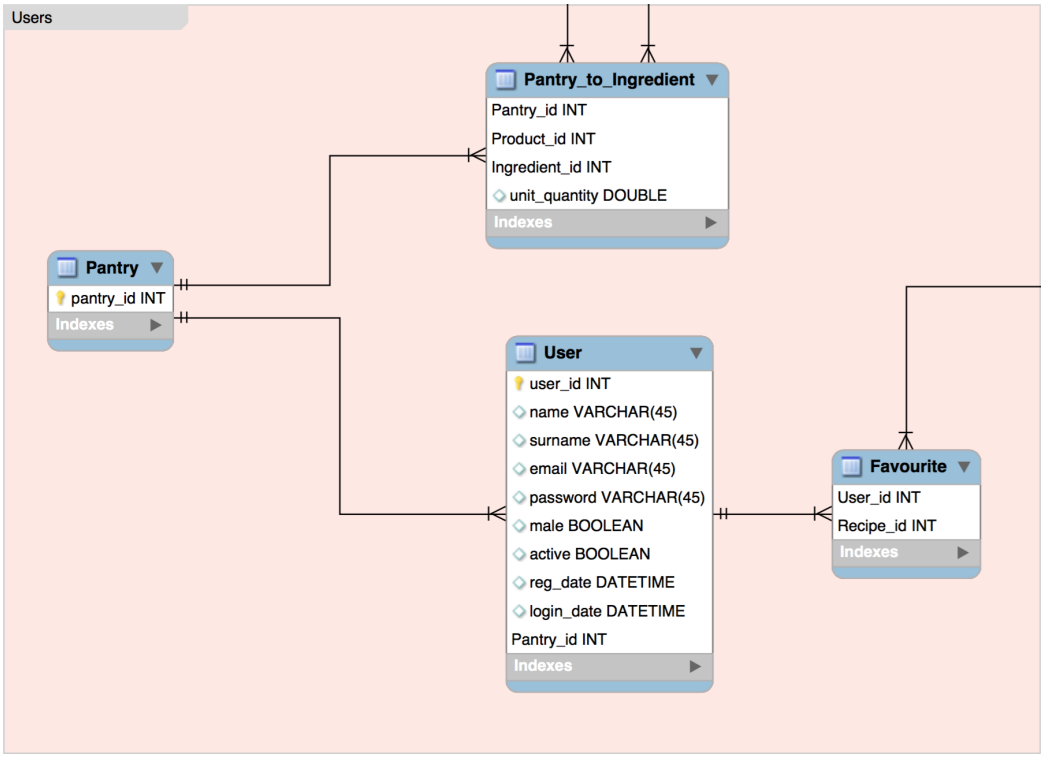
  hrani podatke o receptu.

- **Ingredient_to_Recipe**
  Tabela Ingredient_to_Recipe predstavlja vmesno tabelo med tabelama Ingredient in Recipe. Služi hranjenju količine sestavine izdelka, ki jo potrebujemo pri nekem receptu. Ta podatek hrani v atributu *unit_quantity*.

- **Web_page**
  Tabela Web_page je namenjena hranjenju osnovnih podatkov o spletnem viru, iz katerega pridobivamo recepte. Te podatke hranimo v atributih:
  - *name* (predstavlja ime spletne strani) ter
  - *link* (predstavlja povezavo na spletno stran).

- **Category**
  Tabela Category hrani imena kategorij jedi. Ta podatek hrani v atributu *name*.

Podatke o **uporabnikih** in njihovih shrambah hranimo v tabelah skupine *Users*, ki jo prikazuje Slika 3.7. Na Sliki 3.4 se nahaja v spodnjem sredinskem delu.



Slika 3.7: Prikaz tabel in relacij skupine *Users*.

Skupino sestavljajo naslednje tabele:

- **User**

  Tabela User je namenjena hranjenju podatkov o uporabniku, kot so:

  - *name* (predstavlja ime),
  - *surname* (predstavlja priimek),
  - *email* (predstavlja e-poštni naslov),
  - *password* (predstavlja geslo),

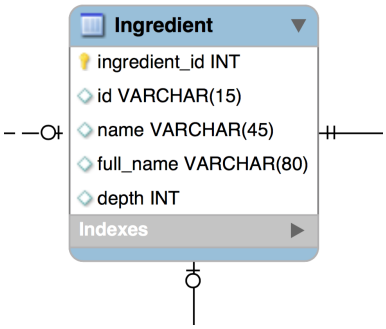
  - *male* (predstavlja logično vrednost, ki definira spol uporabnika (0 - ženska, 1 - moški)),
  - *active* (predstavlja logično vrednost, ki nam da informacijo o profilu uporabnika (0 - neaktiviran, 1 - aktiviran)),
  - *reg_date* (predstavlja datum in čas registracije),
  - *login_date* (predstavlja datum in čas zadnje registracije uporabnika).

- **Pantry**
  Tabela Pantry vsebuje unikaten ključ shrambe, ki ga hrani v atributu *pantry_id*.

- **Pantry_to_Ingredient**
  Tabela Pantry_to_Ingredient je vmesna tabela med tabelo Pantry in tabelo Ingredient. Služi hranjenju količine neke sestavine oziroma izdelka v shrambi. Ta podatek hrani v atributu *unit_quantity*.

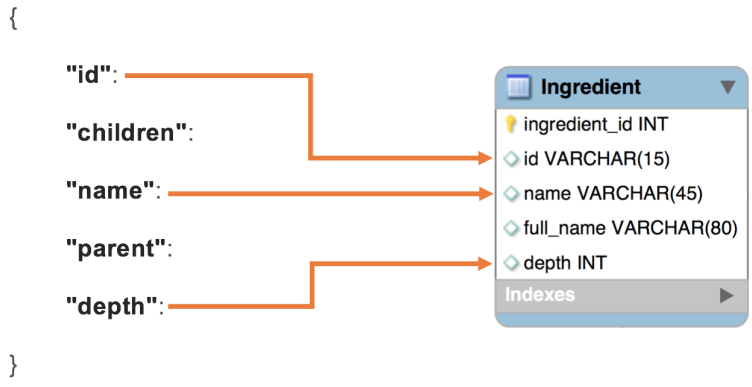
- **Favourite**
  Tabela Favourite je vmesna tabela med tabelo User in Recipe. Namenjena je hranjenju priljubljenih receptov nekega uporabnika.

Srce podatkovne zbirke predstavlja tabela Ingredient, ki jo prikazuje Slika 3.8 in se na Sliki 3.4 nahaja na sredini med posameznimi tremi sklopi.



Slika 3.8: Prikaz osrednje tabele Ingredient.

V tabeli hranimo celotno drevo sestavin, ki smo ga predstavili v podpoglavju 3.1. Slika 3.9 prikazuje pretvorbo iz strukture JSON v tabelo Ingredient. Atributa *children* in *parent* ne hranimo, saj ta podatek pridobimo iz atributa *id*. Atribut *full_name* predstavlja polno ime sestavine, ki ga sestavlja atribut *name* ter imena vseh prednikov do korena (npr. *name*: polnomasten, *full_name*: jogurt - grški - polnomasten).



Slika 3.9: Pretvorba iz strukture JSON v tabelo Ingredient.

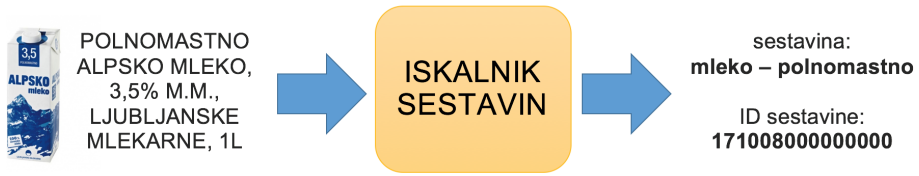
Poglavje 4

Razvoj funkcionalnosti

V tem poglavju bomo predstavili glavne funkcionalnosti avtomatiziranega sistema, APIja, spletne aplikacije in mobilne aplikacije. Najprej bomo predstavili razvoj in delovanje iskalnika sestavin, ki s pomočjo iskalnega algoritma za podan vhod predlaga sestavino. Nadaljevali bomo s predstavitvijo spletnega luščilnika podatkov, ki je namenjen luščenju podatkov o izdelkih ter receptih iz spletnih virov. Pridobljene podatke nato strukturirano shrani v podatkovno zbirko. Sledila bo predstavitev iskalnika receptov, ki na podlagi izdelkov in sestavin, ki jih imamo v shrambi, predlaga recepte, ki jih lahko naredimo. Nato bomo predstavili iskalnik izdelkov na računu, ki na podlagi vhoda, ki je zapis imena izdelka na računu, poišče izdelek v podatkovni zbirki, ki pripada podanem zapisu. V okviru iskalnika izdelkov na računu bomo predstavili tudi algoritem Levenshteinove razdalje, katerega uporablja iskalnik. Nato bomo predstavili API, ki skrbi za povezovanje med spletno aplikacijo, mobilno aplikacijo in strežniškim delom. Na koncu tega poglavja bomo predstavili še razvoj funkcionalnosti spletne aplikacije in mobilne aplikacije.

4.1 Iskalnik sestavin

Iskalnik sestavin je ena od komponent avtomatiziranega sistema, ki teče na strežniškem delu. Glavni namen iskalnika je, da na podlagi vhodnega niza, ki je ime izdelka, predlaga sestavino. Iskalnik je napisan v programskem jeziku Python. Poleg programskega jezika Python iskalnik uporablja vmesnik PyMySQL za dostop do podatkovne zbirke MySQL preko programskega jezika Python. Obe tehnologiji smo predstavili v podpoglavju 2.1. Slika 4.1 prikazuje primer izhoda iskalnika sestavin pri danem vhodu.



Slika 4.1: Prikaz primera vhoda in izhoda iskalnika sestavin.

Psevdo kodo iskalnika sestavin prikazuje Algoritem 3. Ta algoritem na začetku s pomočjo Algoritma 2 (algoritem za gradnjo slovarja) na podlagi vhodnega niza, ki predstavlja ime izdelka, zgradi slovar najdenih sestavin. Algoritem za gradnjo slovarja pretvori vhodni niz v seznam besed. Za vsako od besed seznama, algoritem uporabi Algoritem 1, ki je namenjen iskanju sestavin v drevesu sestavin, ki ga predstavlja tabela Ingredient. Imenu izdelka, ki je vhod algoritma za iskanje sestavin postopoma od zadaj reže znake ter novo nastal niz primerja z imeni sestavin v tabeli Ingredient. Sestavino, katere ime vsebuje novo nastal niz, doda v slovar *ingDict*. To ponavlja, dokler ni razlika med dolžini novo nastalega niza ter originalnim nizom manjša od 0.62 ali pa je novi niz krajši od treh znakov. Izhod algoritma za iskanje sestavin predstavlja slovar *ingDict*, katerega ključ je *id* sestavine. Vrednost ključa pa predstavlja seznam, ki vsebuje ime sestavine, nivo na katerem se nahaja ter skupno število istih znakov, s katero je bila sestavina primerjana.

Algoritem 1: Iskanje sestavin

Input: productStr: niz ime izdelka

Output: ingDict: slovar najdenih sestavin

```
1  Function findIngredients (productStr)
2      stringLen ← len(productStr);
3      i ← stringLen;
       /* Niz režemo od zadaj.                                    */
4      while i > 0 do
5          newString ← left(productStr, i);
6          if ((len(newString) / stringLen) < 0.62) ∨
           (len(newString) < 3) then
7              return ingDict;
8          Pridobi sestavine iz podatkovne zbirke, ki vsebujejo niz
           newString in jih dodaj v slovar ingDict;
9          i−−;
10     return ingDict;
```

Algoritem za gradnjo slovarja se nato sprehodi čez slovar *ingDict* ter za vsako vrednost slovarja izračuna oceno. Oceno posamezne najdene sestavine izračunamo s pomočjo enačbe 4.1.

Naj bo S seznam *productArray*, ki vsebuje besede imena artikla. Posamezne besede, ki se nahajajo na i-tem mestu seznama S označimo z S_i. $|S_i|$ predstavlja dolžino besede S_i.

Naj bo $value_k$ vrednost, ki pripada ključu id_k, kjer k predstavlja pozicijo v slovarju najdenih sestavin *ingredientDict*, ki ga dobimo kot rezultat Algoritma 1. Vrednost $value_k$ predstavlja seznam treh spremenljivk:

- $name_k$ je ime sestavine, $|name_k|$ predstavlja dolžino besede $name_k$,
- $depth_k$ je globina vozlišča, kjer se sestavina nahaja,
- $sameCharNum_k$ je skupno število istih znakov.

Oceno i-te najdene sestavine izračunamo po naslednji enačbi:

$$score_i = \frac{min(|name_k|, |S_i|)}{max(|name_k|, |S_i|)} + \frac{sameCharNum_k}{|name_k|} \tag{4.1}$$

Algoritem 2: Gradnja slovarja

Input: productName: ime izdelka

Output: searchDictionary: slovar sestavin

```
1  Function buildDictionary (productName)
2      searchDictionary ← {};
3      Pretvori vse črke niza productName v male;
4      Odstrani znake: ",", "(", ")"iz niza productName;
5      productArray ← productName.split(" ");
6      foreach productStr in productArray do
           /* Algoritem 1.                                    */
7          ingredientDict ← findIngredients(productStr);
8          foreach id, (name, depth, sameCharNum) in
           ingredientDict.items() do
9              Izračunaj vrednost spremenljivke score po enačbi 4.1.
10             searchDictionary[id] ← [name, depth, score];
11     return searchDictionary;
```

Zgrajeni slovar predstavlja izhod Algoritma 2. Ključ v slovarju predstavlja *id* sestavine. Vrednost ključa pa predstavlja seznam, ki vsebuje ime sestavine, nivo na katerem se nahaja ter oceno. Iskalni algoritem nato zgrajen slovar uporabi tako, da se sprehodi čez urejen seznam ključev slovarja ter za vsak ključ, ki predstavlja *id* sestavine, preveri, če seznam vsebuje njegovega starša. Če seznam vsebuje starša, se vrednosti ključa prišteje ocena starša. Izhod algoritma predstavlja *id* sestavine, ki ima najboljšo oceno.

Algoritem 3: Iskalni algoritem

Input: productName: niz ime izdelka

Output: bestID: id sestavine

```
1  Function searchEngine (productName)
       /* Algoritem 2.                                              */
2      dict ← buildDictionary(productName);
3      bestMark ← 0;
4      bestID ← "";
5      bestDepth ← 5;
6      foreach id in Sortiran seznam dict.keys() do
7          depth ← dict[id][1];
8          mark ← dict[id][2];
           /* Pridobimo ID starša.                                  */
9          pID ← str(left(id, depth * 3)).ljust(15, "0");
           /* Dodamo oceno.                                         */
10         try dict[id][2] = dict[pID][2] + mark;
11         if (dict[id][2] > bestMark) ∨
           ((dict[id][2] == bestMark) ∧ (depth < bestDepth)) then
12             bestMark ← dict[id][2];
13             bestID ← id;
14             bestDepth ← depth;
15     return bestID;
```

4.2 Spletni luščilnik podatkov

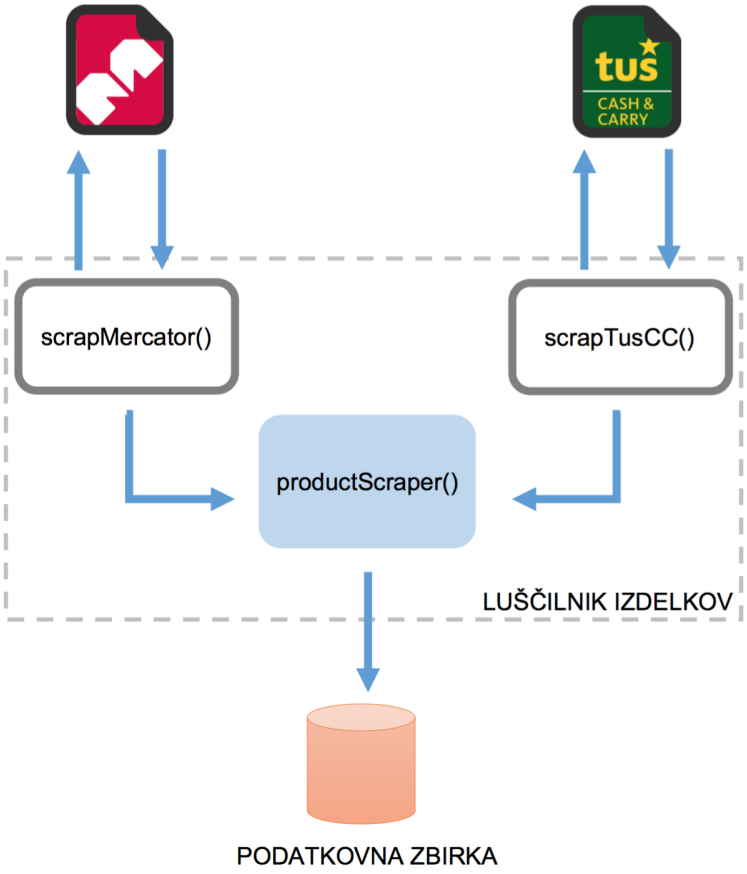
Spletni luščilnik podatkov je namenjen pridobivanju in luščenju podatkov o izdelkih ter receptih iz spletnih virov. Poleg samega luščenja podatkov pa spletni luščilnik izluščene podatke, ki so običajno slabše strukturirani, strukturirano shrani v podatkovni zbirki. Ker potrebujemo podatke o izdelkih in receptih, smo razvili dva spletna luščilnika podatkov:

- luščilnik izdelkov,
- luščilnik receptov.

4.2.1 Luščilnik izdelkov

Luščilnik izdelkov je namenjen luščenju podatkov o izdelkih, ki jih pridobimo iz spletnih virov. Luščilnik vsebuje dva modula. Posamezni modul pridobiva podatke o izdelkih iz posameznega spletnega vira. Pridobljene podatke nato strukturirano shrani v seznam slovarjev, ki ga vrne glavni funkciji luščilnika. Za razvoj luščilnika smo uporabili skriptni jezik Python ter knjižnico Requests, ki je namenjena delu z zahtevki HTTP. Obe tehnologiji smo predstavili v podpoglavju 2.1.

Slika 4.2 prikazuje zasnovo luščilnika izdelkov. Bela pravokotnika (*scrapMercator()* in *scrapTusCC()*) predstavljata modula, medtem ko modri pravokotnik (*productScraper()*) predstavlja glavno funkcijo luščilnika.



Slika 4.2: Zasnova luščilnika izdelkov.

Prvi modul *scrapMercator* pridobiva podatke o izdelkih iz spletne trgovine Mercator [20]. Modul s pomočjo funkcije *requests.get* (Koda 4.1) naredi zahtevek HTTP GET na spletni vir Mercator trgovine.

Koda 4.1: Primer uporabe funkcije *requests.get*.

```
1 data = requests.get(url, params=parameters).json()
```

Rezultat funkcije predstavlja objekt JSON, ki vsebuje podrobne podatke o izdelku. Primer rezultata zgornje funkcije prikazuje Koda 4.2.

Koda 4.2: Rezultat zahtevka spletne trgovine Mercator.

```
>>> data
[
   {
      "url" : "/market/izdelek/14650721/polnomastno-trajno...",
      "itemId" : "14650721",
      "mainImageSrc" :"/market/img/cache/products/4169...",
      "short_name" :"POLNOMASTNO TRAJNO MLEKO, 3,5% M.M.,...",
      "type" : "product",
      "ordNum" : 0,
      "data" : { ... },
      "total" : 17776,
      "className" :"size11"
   }
]
```

Nato modul podatke izlušči ter jih strukturirano shrani v seznam slovarjev, ki ga uporabi kot vhod glavne funkcije luščilnika *productScraper*. Glavno funkcijo luščilnika bomo predstavili v nadaljevanju. Primer vhoda glavne funkcije luščilnika prikazuje Koda 4.3.

Koda 4.3: Primer vhoda glavne funkcije luščilnika.

```
>>> mercatorProducts
[
   {
      "name" :           "POLNOMASTNO ALPSKO MLEKO ...",
      "imgSrc" :         "/market/img/cache/prod...",
      "brand_name" :     "ALPSKO MLEKO",
      "ean" :            "3838800052007",
      "current_price" :  "10.5",
      "price_per_unit" : "0.875",
      "unit_quantity" :  "12",
      "base_unit" :      "l",
      "store" :          "Mercator"
   }, ...
]
```

Drugi modul *scrapTusCC* pa pridobiva podatke o izdelkih iz spletne trgovine Tuš Cash & Carry [34]. Modul s pomočjo funkcije *requests.post* naredi zahtevek HTTP POST na spletni vir Tuš Cash & Carry trgovine. Primer uporabe te funkcije prikazuje Koda 4.4.

Koda 4.4: Primer uporabe funkcije *requests.post*.

```
data = requests.post(url, data=parameters).json
```

Prav tako kot pri zgornji funkciji modula *scrapMercator* je tudi pri tem modulu rezultat zahtevka objekt JSON, katerega v nadaljevanju izluščimo. Primer rezultata prikazuje Koda 4.5.

Koda 4.5: Rezultat zahtevka spletne trgovine Tuš.

```
{
   "docs" : [
      {
         "_id" : "543b0a559785c495304a34f5",
         "Visina" : 6.3,
         "Em_Naziv" : "KOS",
         "search_5" : "instant juhe akcijski katalog",
         "prices" : [ ... ],
         "attr_1005_9" :"JUHE INSTANT -KOCKE",
         "attr_1007_12" :"24",
         "SeznamArtikelPopustov" :[,,, ],
         "page" : "53faf82bfaa8a2b245ba5913",
         "Dolzina" : 15.1,
         "attr_1005_56" :"JUHE INSTANT -KOCKE",
         "url" : "/izdelek/kocka-knorr-kokosja-120g-674173",
         "attr_1000" :"KOKOSJA KOCKA KNORR, 120G",
         "images" : [ ... ],
         "search_10" :"kocka knorr kokosja 120g ...",
         "attr_1039" :"ITALIJA",
         "attr_1007_0" :"24",
         "attr_1013" :"KOKOSJA",
         "Dobavitelji" :[ ... ],
         ...
      }
   ],
   "html" : " ... "
}
```

Nato ta modul, prav tako kot prvi, podatke izlušči ter jih strukturirano shrani v slovar, ki ga poda kot vhod glavne funkcije luščilnika. Primer vhoda glavne funkcije luščilnika predstavlja Koda 4.6.

Koda 4.6: Rezultat zahtevka spletne trgovine TušCC.

```
>>> tusCCProducts
[
   {
      "name" :           "KOKOSJA JUSNA KOCKA, KNORR, 120G",
      "imgSrc" :         "/izdelek/8000...",
      "brand_name" :     "KNORR",
      "ean" :            "8000830301317",
      "current_price" :  "0.9",
      "price_per_unit" : "7.5",
      "unit_quantity" :  "0.12",
      "base_unit" :      "kg",
      "store" :          "TusCC"
   }, ...
]
```

Glavna funkcija luščilnika *productScraper* se nato sprehodi čez seznam slovarjev ter za posamezen izdelek preveri, če se že nahaja v podatkovni zbirki. Ob predpostavki, da se izdelek že nahaja, mu osveži ceno. Če pa izdelka še ni v podatkovni zbirki, funkcija preveri, če se katera od besed imena izdelka nahaja na črni listi (angl. blacklist). Črna lista predstavlja množico besed, ki niso sestavine (npr. sončenje, mačke, psi). Ob predpostavki, da besede ni na črni listi, uporabimo iskalnik sestavin, ki mu kot vhod podamo ime izdelka. Rezultat iskalnika sestavin predstavlja *id* sestavine. Vse pridobljene podatke nato shranimo v tabelo Product. Tabelo Product smo podrobneje predstavili v podpoglavju 3.2.

4.2.2 Luščilnik receptov

Luščilnik receptov je namenjen luščenju podatkov o receptih, ki jih pridobimo iz spletnih virov. Prav tako kot tudi luščilnik izdelkov vsebuje luščilnik receptov module. Naloga modula je pridobivanje podatkov o izdelkih iz posameznega spletnega vira. Podatke nato izlušči ter strukturirano shrani v seznam slovarjev, ki predstavlja vhod glavne funkcije luščilnika. Za razvoj luščilnika smo uporabili skriptni jezik Python, knjižnico Requests ter knjižnico Beautiful Soup. Vsako izmed tehnologij smo predstavili v podpoglavju 2.1.

Modul *scrapKulinarika* pridobiva podatke o receptih iz spletnega portala Kulinarika.net [16]. Modul s pomočjo funkcije *requests.get* naredi zahtevek HTTP GET na spletni vir recepta, ki se nahaja na portalu Kulinarika.net. Rezultat funkcije je vsebina dokumenta HTML spletnega vira. Sledi luščenje vsebine. Kot prikazuje Koda 4.7, s pomočjo knjižnice Beautiful Soup iz vsebine izluščimo sestavine recepta. Funkciji *find_all*, ki je del knjižnice Beautiful Soup, podamo značko (angl. tag) dokumenta HTML, kjer se sestavine nahajajo.

Koda 4.7: Iskanje sestavin v dokumentu HTML.

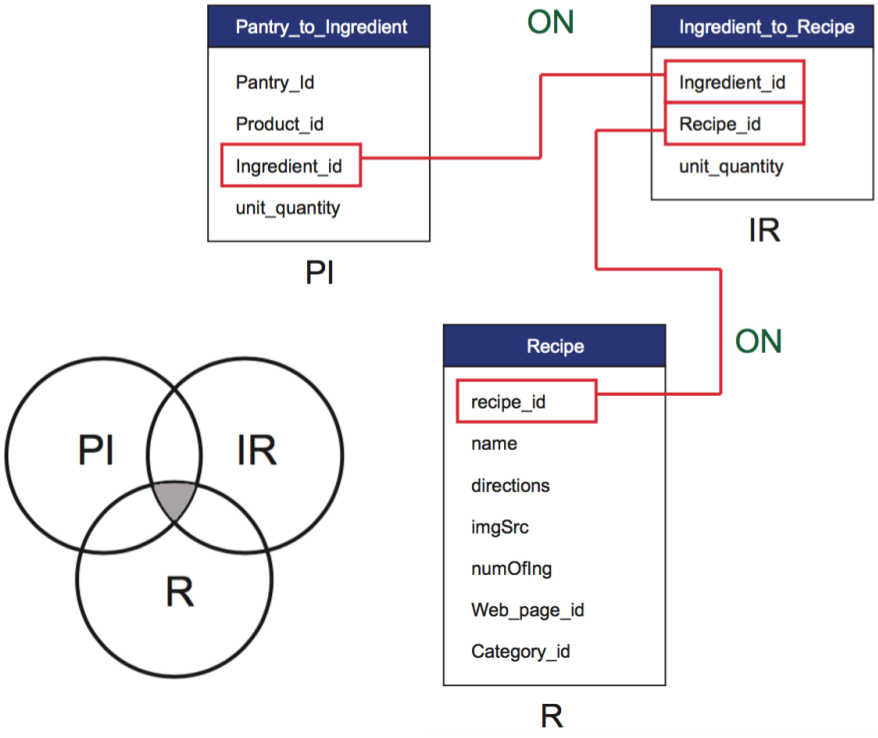
```
1 page = requests.get(url)
2 soup = BeautifulSoup(page.content)
3
4 ingredients = soup.find_all("p", {"itemprop": "ingredients"})
```

Iz recepta pridobljene sestavine nato shranimo v slovar, ki ga podamo kot vhod glavne funkcije luščilnika. Glavna funkcija luščilnika nato nad vsako pridobljeno sestavino uporabi iskalnik sestavin, ki za podan vhod določi sestavino iz drevesa sestavin. Vse pridobljene podatke glavna funkcija shrani v podatkovno zbirko.

4.3 Iskalnik receptov

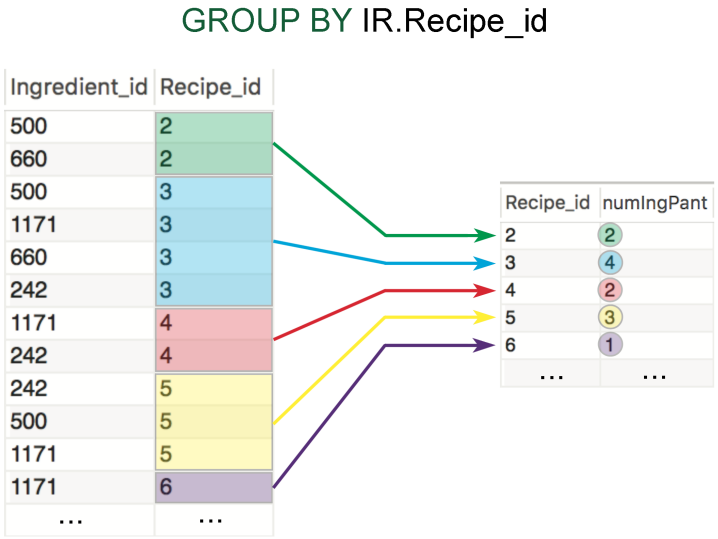
Glavni namen iskalnika receptov je, da nam vrne vse recepte jedi, ki jih lahko naredimo iz sestavin in izdelkov, ki jih imamo v shrambi. Iskalnik receptov uporablja poizvedbeni jezik SQL, s katerim izvaja poizvedbe nad podatki podatkovne zbirke.

Prvi korak iskalne poizvedbe je povezava tabel *PI* (Pantry_to_Ingredient), *IR* (Ingredient_to_Recipe) in *R* (Recipe). Kot prikazuje Slika 4.3, vse tri tabele povežemo s pomočjo pogojnega stika (SQL: INNER JOIN oziroma JOIN), katerega rezultat je presek vseh treh tabel. Tabeli *PI* in *IR* povezuje atribut Ingredient_id, medtem ko tabeli *IR* in *R* povezuje atribut Recipe_id. S pomočjo preseka dobimo sestavine, ki pripadajo tabelama *PI* in *IR*, kar nam omogoča izpis receptov, ki vsebujejo sestavine naše shrambe.



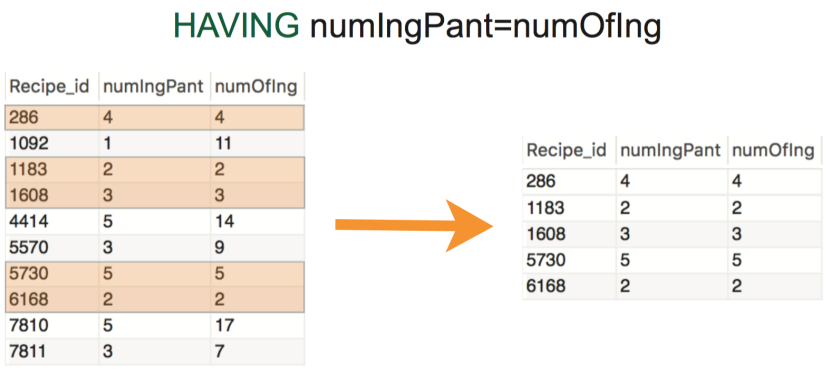
Slika 4.3: Prikaz pogojnega stika med tabelami *PI*, *IR* in *R*.

Naslednji korak iskalne poizvedbe je združitev vrstic (SQL: GROUP BY) po stolpcu IR.Recipe_id. Rezultat združitve vrstic (atribut *numIngPant*) je število vrstic, ki smo jih združili. To predstavlja število sestavin, ki jih imamo za določen recept, kot je prikazano na Sliki 4.4.



Slika 4.4: Rezultat združitve vrstic.

Sledi še primerjanje ter filtriranje atributov *numIngPant* (število sestavin, ki jih imamo za določen recept) ter *numOfIng* (število sestavin, ki jih ima določen recept). Kot prikazuje Slika 4.5, rezultate filtriramo s pomočjo SQL HAVING stavka ob pogoju (numIngPant=numOfIng), kar nam omogoča izpis vseh receptov, ki jih lahko naredimo iz sestavin, ki jih imamo v shrambi.



Slika 4.5: Rezultat filtriranja atributov *numIngPant* in *numOfIng*.

Poizvedbo iskalnika receptov prikazuje Koda 4.8.

Koda 4.8: Poizvedba iskalnika receptov.

```
1  SELECT      IR.Recipe_id,
2              COUNT(*) AS numIngPant,
3              R.numOfIng
4
5  FROM        Pantry_to_Ingredient PI
6              JOIN Ingredient_to_Recipe IR
7                   ON (PI.Ingredient_id=IR.Ingredient_id)
8              JOIN Recipe R
9                   ON (IR.Recipe_id=R.Recipe_id)
10
11 GROUP BY    IR.Recipe_id
12 HAVING      numIngPant=numOfIng
```

4.4 Iskalnik izdelkov na računu

Vhod iskalnika izdelkov na računu predstavlja ime izdelka, ki ga dobimo s pomočjo optičnega prepoznavanja znakov (OCR) na računu. Iskalnik izdelkov s pomočjo iskalnega algoritma nato poišče ustrezen izdelek, ki pripada podanemu vhodu. Ker rezultat optičnega prepoznavanja znakov ni popolnoma natančen, pri primerjanju vhodnega niza z nizi v podatkovni zbirki, uporabimo algoritem Levenshteinove razdalje. Algoritem Levenshteinove razdalje določi podobnost med podanima nizoma. Levenshteinovo razdaljo ter iskalni algoritem predstavljamo v nadaljevanju.

4.4.1 Levenshteinova razdalja

Levenshteinova razdalja (angl. Levenshtein distance), velikokrat imenovana tudi kot urejevalna razdalja (angl. edit distance), je v računalniški znanosti mera za merjenje razlike med dvema nizoma. Ime je dobila po ruskem znanstveniku Vladimirju Levenshteinu, ki je leta 1965 razvil algoritem. Levenshteinova razdalja dveh nizov predstavlja najmanjše število operacij, ki jih moramo narediti, da iz prvega niza dobimo drugega [21]. Te operacije so:

- **vrivanje** znaka v niz,
- **zamenjava** znaka v nizu z drugim ter
- **brisanje** znaka iz niza.

Naj bo a prvi niz in b drugi. Levenshteinovo razdaljo med nizoma a in b označimo z $LR(a, b)$. Večja kot je Levenshteinova razdalja, manj sta si niza podobna, saj za preoblikovanje niza a v b porabimo več operacij (vrivanj, zamenjav, brisanj).

Primer:

- za $a = SESTAVINE$ ter $b = SESTAVINE$ je $LR(a, b) = 0$, saj sta niza a in b popolnoma enaka in posledično ne potrebujemo nobene od operacij
- za $a = MATEJ$ ter $b = MATEJA$ je $LR(a, b) = 1$, saj potrebujemo operacijo vrivanja (A) na koncu niza a, da ga spremenimo v niz b
- za $a = MOKA$ ter $b = KODA$ je $LR(a, b) = 2$, saj potrebujemo dve operaciji zamenjave $(M \rightarrow K)$ ter $(K \rightarrow D)$

Pri reševanju problema lahko uporabljamo pristop rekurzivne rešitve ali pa pristop dinamičnega programiranja. V diplomski nalogi smo uporabili implementacijo Levenshteinove razdalje po pristopu dinamičnega programiranja, saj algoritem s pomočjo matrike, ki sproti hrani in uporablja že izračunane vmesne rešitve, ogromno prihrani na časovni zahtevnosti. S tem se izognemo rekurzivnim klicem, ki bi že izračunan rezultat računali večkrat in bi s tem algoritem privedli do eksponentne časovne zahtevnosti.

Opis delovanja

Naj bo a prvi niz ter b drugi niz. Dolžino niza a označimo z $|a|$, dolžino niza b pa z $|b|$. Posamezni i-ti znak niza a označimo z a_i, pri čemer $i = 1...|a|$. Posamezni j-ti znak niza b pa označimo z b_j, pri čemer $j = 1...|b|$. M naj bo matrika vmesnih rešitev velikosti $(|a| + 1) \times (|b| + 1)$.
Velja:

- $M[0, 0] = 0$
- $M[i, 0] = i$, pri čemer $i = 1...|a|$
- $M[0, j] = j$, pri čemer $j = 1...|b|$

Vse ostale elemente v matriki pa izračunamo po naslednji enačbi:

$$M[i,j] = \min \begin{cases} M[i-1,j] + 1 \\ M[i,j-1] + 1 \\ M[i-1,j-1] + 1_{(a_i \neq b_j)} \end{cases} \tag{4.2}$$

pri čemer $1_{(a_i \neq b_j)}$ predstavlja karakteristično funkcijo, ki je definirana kot:

$$1_{(a_i \neq b_j)} = \begin{cases} 0 & \text{če } a_i = b_j, \\ 1 & \text{drugače.} \end{cases} \tag{4.3}$$

V enačbi 4.2 prvi element minimuma predstavlja operacijo brisanja, drugi operacijo vrivanja tretji pa operacijo zamenjave [30]. Končno rešitev Levenshteinove razdalje predstavlja zadnji element matrike, ki ga najdemo na mestu $M[|a|, |b|]$.

Algoritem 4 za izračun Levenshteinove razdalje na začetku definira dvodimenzionalno tabelo M, ki je namenjena shranjevanju vmesnih rezultatov. Nato sledi korak določanja prvih vrednosti vrstic ter stolpca matrike M. Glavni korak algoritma se zgodi v vrsticah (9-17), kjer določimo vrednost $M[i, j]$, ki je minimalna vrednost ene od operacij. Rezultat algoritma predstavlja minimalno število korakov, ki jih potrebujemo, da niz a pretvorimo v niz b. Časovna zahtevnost algoritma je $O(|a| \times |b|)$.

Algoritem 4: Levenshteinova razdalja - dinamično programiranje [17]

Input: a, b : niza a in b

Output: $M[|a|][|b|]$: minimalno število korakov

```
1  Function LevenshteinDistance(a, b)
2      Definiramo dvodimenzionalno tabelo M, ki je velikosti
       (|a| + 1) × (|b| + 1).
3      for i ← 1 to |a| do
4          M[i, 0] = i;
5      for j ← 1 to |b| do
6          M[0, j] = j;
7      for i ← 1 to |a| do
8          for j ← 1 to |b| do
9              if (a[i] == b[j]) then
10                 cost = 0;
11             else
12                 cost = 1;
13             M[i, j] = min(
14                             M[i − 1, j − 1] + cost,
15                             M[i − 1, j] + 1,
16                             M[i, j − 1] + 1
17                           );
18     return M[|a|, |b|];
```

4.4.2 Iskalni algoritem

Iskalni algoritem na podlagi vhoda, ki je zapis izdelka na računu, pridobljen s pomočjo OCR, ugotovi kateremu izdelku v podatkovni zbirki pripada.

Algoritem 5 za iskanje izdelka kot vhod sprejme spremenljivko *billRecordOCR*, ki predstavlja zapis izdelka na računu, pridobljen s pomočjo OCRja. Algoritem na začetku pridobi podatke o vseh zapisih izdelkov z računov (*record_name*) ter ključnih izdelkov, ki pripadajo tem zapisu (*Product_id*), pri čemer uporabi pogojni stik med tabelama Bill in Product_to_Store. Vse pridobljene vrednosti nato shrani v slovar *billRecords*, pri čemer je ključ slovarja *Product_id*, pripadajoča vrednost temu ključu pa zapis *record_name*. Nato se algoritem sprehodi čez slovar *billRecords* ter vsako vrednost slovarja primerja z vhodom algoritma. Pri tem uporabi prej predstavljen algoritem Levenshteinove razdalje, ki ugotovi minimalno število operacij potrebnih za to, da vhodni zapis pretvorimo v zapis iz trenutne vrednosti slovarja. Izdelek, za katerega potrebujemo najmanj operacij, si zapomni v spremenljivko *bestID*. Na koncu algoritem primerja število operacij pretvorbe in število, ki predstavlja polovico števila vseh znakov vhodnega zapisa. Ob predpostavki, da je število operacij manjše, vrne id izdelka. Če pa je število operacij večje, pa vrnemo negativno vrednost (-1), kar se obravnava kot, da za podan vhod ni bilo najdenega izdelka.

Algoritem 5: Iskalni algoritem

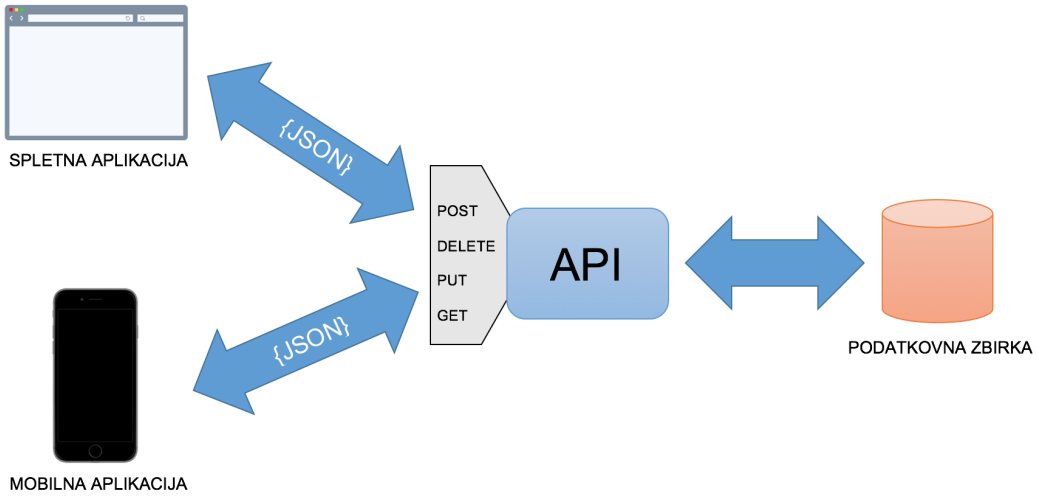
Input: billRecordOCR : zapis izdelka na računu

Output: productID ∨ -1 : id izdelka ∨ -1

```
1  Function findProduct(billRecordOCR)
2      Združi tabeli Bill in Product_to_Store s pomočjo pogojnega stika
       ter pridobi vse zapise record_name in Product_id. Rezultat shrani v
       slovar billRecords, kjer je posamezen ključ Product_id in vrednost
       record_name.
3      bestPrice ← ∞;
4      bestID ← -1;
5      foreach id, billRecord in billRecords.items() do
           /* Algoritem 4.                                              */
6          price ← LevenshteinDistance(billRecordOCR, billRecord);
7          if (price < bestPrice) then
8              bestPrice ← price;
9              bestID ← id;
10     if (bestPrice < billRecordOCR.length/2) then
11         return bestID;
12     return -1;
```

4.5 Aplikacijski programski vmesnik (API)

API (angl. Application Programming Interface) je namenjen kot vmesnik za komunikacijo med odjemalci (spletna aplikacija, mobilna aplikacija) ter strežniškim delom. Vsebuje nabor funkcij, s katerimi se izvaja interakcija med odjemalci ter strežniškim delom. Zgrajen je po arhitekturnem slogu REST, ki predstavlja dobro prakso pri načrtovanju sodobnih spletnih storitev [18]. API je napisan v programskem jeziku PHP, za komuniciranje s podatkovno zbirko pa uporablja vgrajeno PHP knjižnico PDO. Programski jezik PHP smo podrobneje predstavili v podpoglavju 2.1. Slika 4.6 prikazuje zasnovo razvitega REST APIja.



Slika 4.6: Zasnova razvitega REST APIja.

4.5.1 Enolični naslovi virov

REST API za naslavljanje virov uporablja URI (angl. Uniform Resource Identifier). URI predstavlja niz znakov, ki se uporablja kot identifikacija vira na spletu. Preko metod dostopa HTTP, FTP in drugih omogoča dostop do virov [35].

Enolične naslove virov APIja smo definirali na podlagi funkcionalnosti sistema ter skupin in tabel podatkovne zbirke. Posamezne hierarhične odvisnosti med viri smo ločili s poševnico (/). Tabela 4.1 prikazuje primer enoličnih virov za dostop do podatkov shrambe.

Podatki	**URI**
Sestavine in izdelki v shrambi	/pantry
Sestavine v shrambi	/pantry/ingredients
Izdelki v shrambi	/pantry/products

Tabela 4.1: Enolični viri za dostop do podatkov o shrambi.

4.5.2 Zahtevki HTTP

REST API uporablja metode HTTP, ki so zasnovane na funkcijah CRUD. CRUD predstavlja ustvarjanje (angl. create), branje (angl. read), posodabljanje (angl. update) in brisanje (angl. delete). Tabela 4.2 prikazuje primerjavo med funkcijami CRUD, operacijami SQL ter metodami HTTP.

CRUD	**SQL**	**HTTP**
Create	INSERT	POST
Read	SELECT	GET
Update	UPDATE	PUT
Delete	DELETE	DELETE

Tabela 4.2: Primerjava CRUD, SQL in HTTP.

S pomočjo zahtevka HTTP POST, vstavimo nov zapis v podatkovno zbirko. Zahtevek HTTP GET nam vrne podatke o instanci oziroma zbirki nekega vira. Zahtevek HTTP PUT nam že obstoječe podatke v zbirki osveži z novimi, ki jih podamo. Zahtevek HTTP DELETE pa izbriše podatek iz podatkovne zbirke.

4.5.3 Statusi HTTP

Pri APIju smo definirali odzivne statuse HTTP, ki omogočajo odjemalcu, da za določen zahtevek ugotoviti, če je bila akcija zahtevka uspešna oziroma neuspešna. Tabela 4.3 prikazuje pomen statusov HTTP našega sistema.

Status HTTP	**Pomen**
200 OK	Uspešno pridobljen vir.
201 CREATED	Uspešen zapis v podatkovni zbirki.
202 ACCEPTED	Uspešno obdelan zahtevek HTTP.
204 NO CONTENT	Uspešno izbrisan podatek iz podatkovne zbirke.
400 BAD REQUEST	Zahteva vsebuje neveljavne podatke.
401 UNAUTHORIZED	Neuspešna avtorizacija.
403 FORBIDDEN	Dostop zavrnjen.
404 NOT FOUND	Vira zahtevka ni bilo mogoče najti.

Tabela 4.3: Pomen statusov HTTP.

4.5.4 Format za izmenjavo podatkov

Za izmenjavo podatkov med APIjem ter odjemalcem uporabljamo JSON. JSON omogoča široko podporo ter je dejansko standard pri večini APIjev spletnih storitev.

Koda 4.9 prikazuje primer zahtevka HTTP POST, pri katerem dodamo nov priljubljen recept. Telo zahtevka vsebuje podatek o ključu izdelka ter avtentifikacijskem žetonu. Poslane podatke v odgovoru strežnika na zgornji zahtevek prikazuje Koda 4.10.

Koda 4.9: Primer zahtevka HTTP POST.

```
HTTP metoda: POST
URI: /recipes/favourites
Telo zahtevka:
{
   id: 67,
   authToken: "..."
}
```

Koda 4.10: Primer poslanih podatkov v odgovoru.

```
{
  meta: {
    message: "Neveljavni podatki"
  },
  data: {}
}
```

4.5.5 Razredi

Glavna razreda APIja predstavljata razreda:

- **ApiDAO**
  Razred vsebuje nabor funkcij, ki s pomočjo poizvedovalnega jezika SQL izvajajo ustrezne operacije nad podatkovno zbirko. Poleg osnovnih funkcij za interakcijo s podatkovno zbirko uporablja tudi *iskalnik receptov* (podpoglavje 4.3) ter *iskalnik izdelkov na računu* (podpoglavje 4.4).

- **ApiResources**
  Razred za določen vir uporabi ustrezno funkcijo razreda ApiDAO ter na ta način pridobi zahtevane podatke, ki jih odjemalec prejme v odgovoru.

4.6 Spletna aplikacija

Spletna aplikacija omogoča različne funkcionalnosti, kot so:

- registracija in prijava,
- pregled/dodajanje/odstranjevanje izdelkov in sestavin v shrambi,
- pregled receptov, ki jih lahko naredimo iz izdelkov in sestavin v shrambi,
- pregled/dodajanje/odstranjevanje priljubljenih receptov.

S pomočjo JavaScript funkcije *requestAPI*, ki jo prikazuje Koda 4.11, spletna aplikacija asinhrono komunicira z APIjem. Pri tem uporablja funkcijo knjižnice jQuery *$.ajax*, ki poskrbi za asinhrono pošiljanje zahtevkov HTTP ter prejemanje odgovorov. Obe uporabljeni tehnologiji smo predstavili v podpoglavju 2.2. Funkcija kot vhod sprejme naslednje parametre:

- *requestURI* predstavlja enoličen naslov vira,
- *requestType* predstavlja metodo HTTP,
- *requestData* predstavlja podatke v strukturi JSON, ki se pošljejo ob zahtevku.

Koda 4.11: Funkcija requestAPI.

```
function requestAPI(requestURI, requestType, requestData) {
   $.ajax({
      url: requestURI,
      type: requestType,
      data: requestData,
      dataType: "json",

      success: function (data, textStatus, jqXHR) {
         return data;
      },

      error: function (jqXHR, textStatus, errorThrown) {
         return textStatus;
      }
   });
}
```

4.6.1 Registracija in prijava

Registracija uporabnika poteka tako, da se po potrditvi obrazca, ki ga uporabnik izpolni, pošlje zahtevek s podatki uporabnika na URI */users*. Koda 4.12 prikazuje primer zahtevka HTTP za registracijo.

Koda 4.12: Primer zahtevka HTTP za registracijo.

```
requestAPI("/users", "POST",
           {
              name: "Matej",
              surname: "Pecnik",
              email: "matej@pecnik.si",
              password: "..."
           }
);
```

Pri **prijavi** v sistem se uporabnik avtenticira s svojo elektronsko pošto in geslom. Pri avtentikaciji se podatki o elektronski pošti ter geslu pošljejo na API, kjer se preveri skladnost podatkov. Če se podatki ujemajo s tistimi v podatkovni zbirki, se uporabniku pošlje avtentikacijski žeton, s katerim dostopa do svojih vsebin.

4.6.2 Pregled, dodajanje in odstranjevanje izdelkov ter sestavin v shrambi

Pri pregledu, dodajanju in odstranjevanju izdelkov in sestavin v shrambi spletna aplikacija zahtevke pošilja na URI */pantry*.

Kot prikazuje Koda 4.13, pri **pregledu** izdelkov in sestavin uporabimo tip zahtevka GET, pri katerem zraven pošljemo še avtentikacijski žeton, ki predstavlja žeton, s katerim se uporabnik identificira ter avtenticira.

Koda 4.13: Primer zahtevka HTTP za pregled izdekov in sestavin shrambe.

```
requestAPI("/pantry", "GET", {authToken: "..."});
```

Dodajanje je možno na tri načine:

- dodajanje izdelkov po imenu izdelka,
- dodajanje izdelkov po črtni kodi,
- dodajanje sestavine po imenu sestavine.

Vsak od načinov dodajanja uporablja iskalni zahtevek, ki na podlagi vhodnega niza vrne *id* izdelka oziroma sestavine. Pridobljen *id* nato uporabimo pri zahtevku za dodajanje.

Brisanje poteka podobno kot dodajanje, le da namesto tipa zahtevka POST uporabimo DELETE, s čimer APIju povemo, da gre za brisanje. Koda 4.14 prikazuje primer brisanja sestavine.

Koda 4.14: Primer zahtevka HTTP za brisanje sestavine iz shrambe.

```
requestAPI("/pantry", "DELETE",
           {
              id: 12,
              ingredient: true,
              authToken: "..."
           }
);
```

4.6.3 Pregled receptov, ki jih lahko naredimo iz izdelkov in sestavin v shrambi

Kot prikazuje Koda 4.15, za pregled receptov, ki jih lahko naredimo, uporabimo zahtevek HTTP GET na URI */recipes*.

Koda 4.15: Primer zahtevka HTTP za pregled receptov.

```
1 requestAPI("/recipes", "GET", {authToken: "..."});
```

4.6.4 Pregled, dodajanje in odstranjevanje priljubljenih receptov

Pregled, dodajanje in odstranjevanje priljubljenih receptov izvedemo s pomočjo zahtevka na URI */recipes/favourites*. Kot prikazuje Koda 4.16, priljubljene recepte pridobimo s pomočjo metode HTTP GET ter uporabniškega avtentifikacijskega žetona.

Koda 4.16: Primer zahtevka HTTP za pregled priljubljenih receptov.

```
1 requestAPI("/recipes/favourites", "GET",
2           {authToken: "..."});
```

Kot prikazuje Koda 4.17, pri dodajanju poleg avtentifikacijskega žetona pošljemo še *id* recepta, ki ga želimo dodati med priljubljene. Brisanje poteka identično kot dodajanje, le da namesto metode HTTP POST v zahtevku pošljemo DELETE.

Koda 4.17: Primer zahtevka HTTP za dodajanje recepta med priljubljene.

```
1 requestAPI("/recipes/favourites", "POST",
2           {id: 261, authToken: "..."});
```

4.7 Mobilna aplikacija

Razvoj mobilne aplikacije je potekal v integriranem razvojnem okolju Xcode. Mobilna aplikacija je napisana v programskem jeziku Swift. Omenjeni tehnologiji smo predstavili v podpoglavju 2.3.

Poleg vseh funkcionalnosti, ki jih nudi spletna aplikacija, nudi mobilna aplikacija še:

- dodajanje izdelkov po črtni kodi ter
- dodajanje izdelkov po računu.

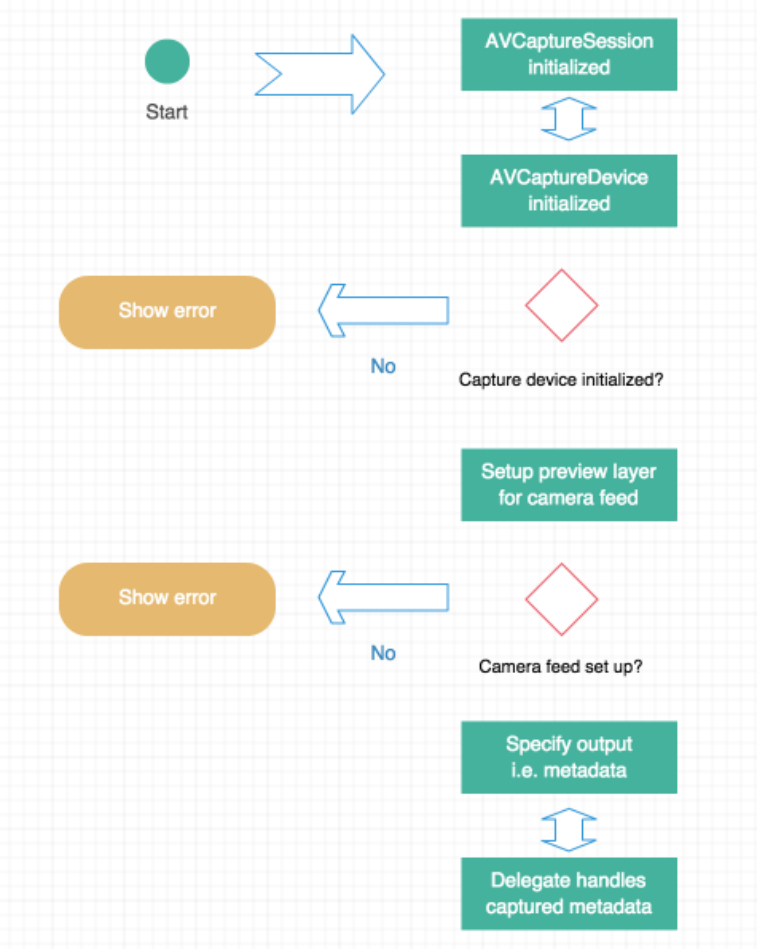
4.7.1 Dodajanje izdelkov po črtni kodi

Ker ima vsak kupljen izdelek v trgovini svojo črtno kodo, ki omogoča hitro in zanesljivo identifikacijo izdelka, smo razvili funkcionalnost, ki nam omogoča, da izdelek identificiramo s pomočjo črtne kode ter ga dodamo v shrambo. Pri razvoju funkcionalnosti smo uporabili naslednje komponente knjižnice AVFoundation:

- **AVCaptureSession**
  AVCaptureSession je objekt, ki koordinira pretok podatkov, katere pridobiva iz avdio ali video vhoda naprave [4].
- **AVCaptureDevice**
  AVCaptureDevice je objekt, ki predstavlja fizično napravo za zajem (npr. kamera) ter lastnosti, ki so s to napravo povezane [3]. S pomočjo tega objekta zajamemo vhodne podatke, ki jih nato predamo inicializirani instanci razreda AVCaptureSession.
- **AVCaptureVideoPreviewLayer**
  AVCaptureVideoPreviewLayer je podrazred razreda CALayer, ki se uporablja za prikaz videa, zajetega s pomočjo naprave za zajem [5]. V našem primeru je to instanca razreda AVCaptureDevice.

- **AVCaptureMetaDataOutput**
  AVCaptureMetaDataOutput je razred, ki prestreže zajete objekte metapodatkov. Te objekte posreduje delegatu za vnaprejšnjo obdelavo. Instanco tega razreda lahko uporabimo za procesiranje specifičnega tipa metapodatka, pridobljenega iz vhodnega podatka [6].

Kot prikazuje diagram poteka na Sliki 4.7, je prvi korak inicializacija razredov AVCaptureSession (začetek seje za zajem) ter AVCaptureDevice. Če naprava za zajem nima pravic oziroma ni primerna za zajem vhoda, inicializacija razreda AVCaptureDevice spodleti, pri čemer se sproži napaka. Ob predpostavki, da je inicializacija uspešna, sledi naslednji korak namestitve plasti za predogled (angl. preview layer) vhodnega vira. V tem koraku je naša naloga, da dodamo plast, ki ji določimo okvir ter razmerje. Naslednji korak je specifikacija izhoda. V tem koraku ustvarimo instanco razreda AVCaptureMetadataOutput ter jo dodamo naši seji za zajem. Nato objektu metadataObjectTypes, določimo nabor metapodatkov, ki jih želimo uporabljati pri zajemu.



Slika 4.7: Diagram poteka zajema ter obdelave črtne kode [8].

Kot prikazuje Koda 4.18, uporabimo metapodatke tipa EAN13 ter EAN8, saj sta ta tipa črtne kode trenutno na tržišču najbolj v uporabi. EAN predstavlja kratico za evropsko številčenje proizvodov (angl. European Article Numbering). Sledi nastavitev razreda kontrolnega pogleda (angl. view controller) aplikacije kot delegata. S tem pridobi razred kontrolnega pogleda dovoljenje za obdelavo metapodatkov.

Koda 4.18: Določitev nabora metapodatkov.

```
metadata.metadataObjectTypes = [AVMetadataObjectTypeEAN8Code,
                                AVMetadataObjectTypeEAN13Code]
```

Nato pridobljene metapodatke dekodiramo ter iz njih izluščimo podatke o črtni kodi, kar prikazuje Koda 4.19. Podatek o črtni kodi uporabimo kot vhod funkcije getProduct. Funkcija s pomočjo zahtevka HTTP GET na API (URI: */pantry/products*) pridobi ime in sliko izdelka, ki mu pripada črtna koda. Izdelek v shrambo dodamo z zahtevkom HTTP POST na API (URI */pantry/products*).

Koda 4.19: Pridobivanje črtne kode iz metapodatkov.

```
for metaData in metadataObjects {
   let decodedData:AVMetadataMachineReadableCodeObject =
      metaData as! AVMetadataMachineReadableCodeObject
   self.eanCode.text = decodedData.stringValue
   self.getProduct(decodedData.stringValue)
}
```

4.7.2 Dodajanje izdelkov po računu

Ker je polnjenje shrambe pri večjih nakupih zamudno, smo razvili funkcionalnost, ki nam omogoča, da izdelke kupljene v trgovini, dodamo s pomočjo skeniranja računa, ki smo ga prejeli ob nakupu.

Dodajanje poteka v štirih korakih:

1. zajem računa,
2. določitev območja izdelkov na računu,
3. optično prepoznavanje znakov in

4. pridobivanje ter dodajanje izdelkov.

Pri **zajemu računa** uporabnik izbere vir, s katerim zajame ali pridobi sliko računa. Kot prikazuje Koda 4.20, ima na voljo zajem računa preko vgrajene kamere naprave ali pa izbiro že zajetega računa iz foto knjižnice, ki se nahaja na mobilni napravi.

Koda 4.20: Izbira vira slike.

```
switch theImageSource {
   case .Camera:
        imagePicker.sourceType =
           UIImagePickerControllerSourceType.Camera
   case .PhotoLibrary:
        imagePicker.sourceType =
           UIImagePickerControllerSourceType.SavedPhotosAlbum
}
```

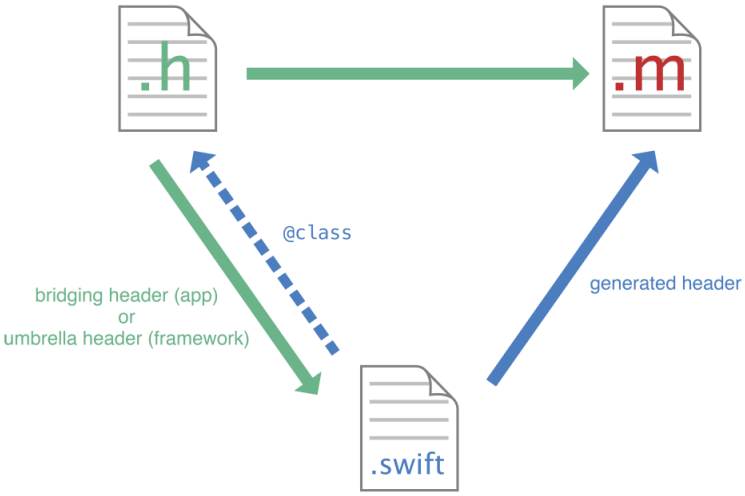
Pridobljeno sliko računa nato prikažemo na pogledu (angl. view) uporabniškega vmesnika, ki služi za vnaprejšnjo obdelavo slike.

Nato s pomočjo okvirja, ki ga ustvarimo s potegom prsta po pogledu, **določimo območje izdelkov** na sliki računa. Pri tem delu uporabimo razreda UITapGestureRecognizer ter UIPanGestureRecognizer. Oba sta del razreda UIGestureRecognizer, katerega naloga je prepoznavanje različnih gest. Prvi razred je namenjen delu z gesto dotika (angl. tap), drugi pa delu z gesto povlečenja (angl. pan ali drag). Razred UIBezierPath poskrbi za izris črt, medtem ko s pomočjo razreda UIImage označen del obrežemo.

Sledi **optično prepoznavanje znakov** (angl. Optical Character Recognition) iz obrezanega dela slike. Pri tem delu smo uporabili sistem za optično prepoznavanje znakov Tesseract, ki smo ga predstavili v podpoglavju 2.2. Ker je ogrodje Tesseract za operacijski sistem iOS napisan v

programskem jeziku Objective-C, naša aplikacija pa v programskem jeziku Swift, je bila naša naloga gradnja premostitvene glave (angl. bridging header) Objective-C .Premostitvena glava Objective-C omogoča, da v programskem jeziku Swift uporabljamo razrede, ki so napisani v programskem jeziku Objective-C.

Kot lahko vidimo na Sliki 4.8, to storimo s pomočjo kreirane zaglavne datoteke (angl. header file), katero uporabljajo tako razredi Objective-C kot tudi Swift. V integriranem razvojnem okolju Xcode ustvarimo novo zaglavno datoteko ter v njej definiramo vse razrede Objective-C, ki jih želimo premostiti. Vsebino zaglavne datoteke naše aplikacije prikazuje Koda 4.21.



Slika 4.8: Premostitvena glava Objective-C [31].

Koda 4.21: Vsebina zaglavne datoteke *Shramba-Bridging-Header.h*.

```
1 #import <TesseractOCR/TesseractOCR.h>
```

Po uspešni namestitvi zaglavne datoteke sledi uporaba teh razredov.

Kot prikazuje Koda 4.22, sprva inicializiramo razred G8Tesseract. Sledi določitev vrednosti spremenljivk inicializiranega razreda, ki se bodo uporabile pri samem algoritmu prepoznave. Te spremenljivke so:

- *language*
  Ta spremenljivka predstavlja jezikovno datoteko .traineddata, ki jo Tesseract uporabi. V našem primeru bo uporabil jezikovno datoteko slv.traineddata, ki vsebuje podatke za prepoznavo slovenskih znakov.

- *engineMode*
  Ta spremenljivka predstavlja način obdelave OCR. Na voljo imamo tri načine:

  - .TesseractOnly uporablja jezikovno datoteko .traineddata (najhitrejši, vendar najmanj natančen)
  - .CubeOnly uporablja jezikovne datoteke .cube (počasnejši, a bolj natančen saj uporablja naprednejše algoritme umetne inteligence)
  - .TesseractCubeCombined (najpočasnejši, vendar najbolj natančen saj vsebuje oba zgornja načina)

  V našem primeru uporabimo .TesseractOnly saj imamo za slovenski jezik na voljo samo jezikovno datoteko slv.traineddata.

- *pageSegmentationMode*
  Ta spremenljivka predstavlja način deljenja besedila v sliki. V našem primeru uporabimo .Avto, ki omogoča avtomatsko segmentacijo strani in s tem sposobnost prepoznavanja prelomov odstavka.

- *maximumRecognitionTime*
  Ta spremenljivka predstavlja časovno omejitev algoritma prepoznave.

- *image*
  Ta spremenljivka predstavlja našo vhodno sliko.

Pri sliki image uporabimo Tesseractov vgrajen filter g8_blackAndWhite. Postopek optične prepoznave sprožimo s klicem funkcije recognize, katera rezultat shrani v spremenljivko recognizedText [33].

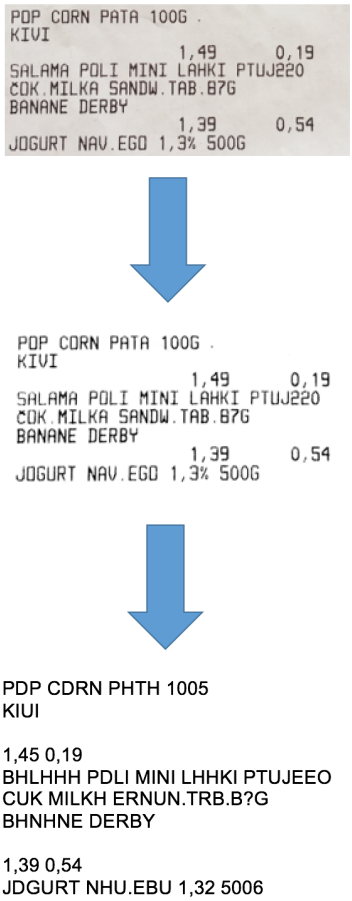
Koda 4.22: Primer uporabe ogrodja Tesseract v aplikaciji.

```
let tesseract = G8Tesseract()

tesseract.language = "slv"
tesseract.engineMode = .TesseractOnly
tesseract.pageSegmentationMode = .Auto
tesseract.maximumRecognitionTime = 60.0
tesseract.image = image.g8_blackAndWhite()
tesseract.recognize()

let tesTXT = tesseract.recognizedText
```

Na Sliki 4.9 so prikazani koraki pri obdelavi računa. Zgornji del slike prikazuje primer vhodne slike, ki se uporabi pri sami obdelavi. Srednji del slike predstavlja rezultat vgrajenega Tesseractovega filtra (g8_blackAndWhite) nad vhodno sliko. Spodnji del slike pa predstavlja končni rezultat OCRja nad srednjo sliko.



Slika 4.9: Koraki pri obdelavi računa. Zgoraj je prikazana vhodna slika, na sredini je slika po filtriranju, medtem ko je spodaj prikazan rezultat optičnega prepoznavanja znakov.

Zadnji korak je **pridobivanje in dodajanje izdelkov**. Pridobljen niz z računa pošljemo na API (URI: */products/bill*). Pri tem zahtevku API uporabi iskalnik izdelkov na računu, ki poišče izdelke ter jih vrne. Iskalnik izdelkov na računu smo podrobneje predstavili v podpoglavju 4.4. Izdelke dodamo s pomočjo zahtevka HTTP POST na API (URI: */pantry/products*).

Poglavje 5

Predstavitev uporabe

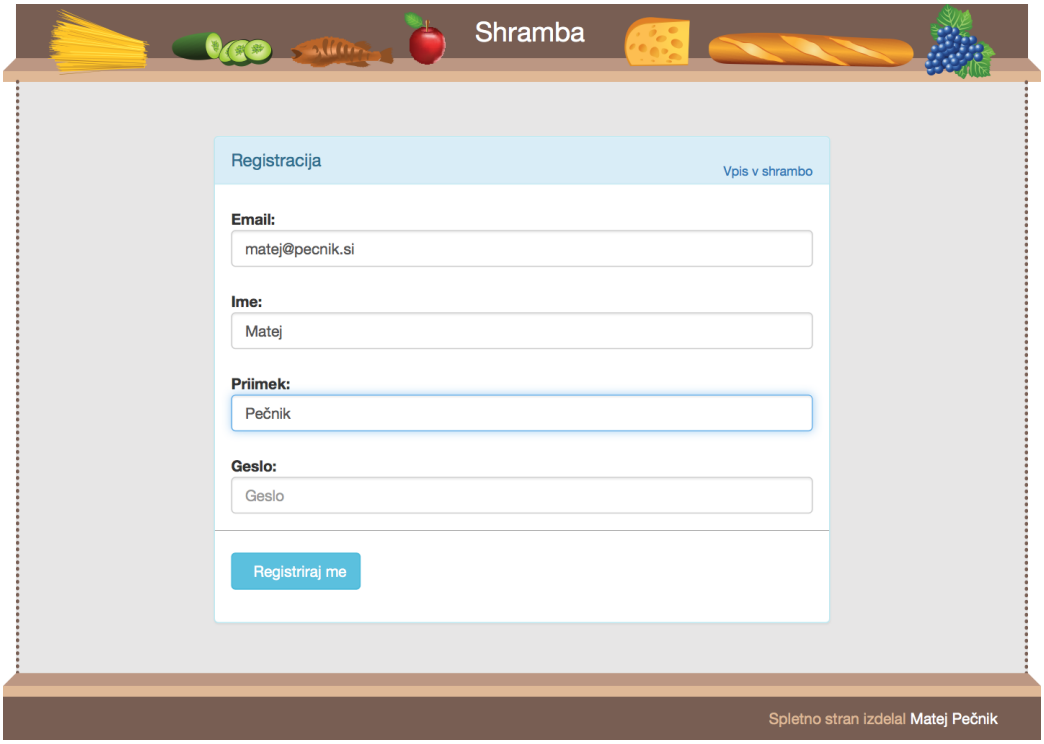
V tem poglavju bomo predstavili uporabo spletne aplikacije in mobilne aplikacije. Začeli bomo s predstavitvijo vmesnika spletne aplikacije, ki je namenjen registraciji oziroma prijavi uporabnika. Nadaljevali bomo s pregledom uporabniškega vmesnika. Na koncu poglavja sledi še predstavitev uporabe mobilne aplikacije.

5.1 Spletna aplikacija

Najprej predstavljamo spletno aplikacijo, ki je sestavljena iz vmesnika za registracijo in prijavo ter spletnih strani, ki omogočajo delo s shrambo ter iskanje in pregledovanje receptov.

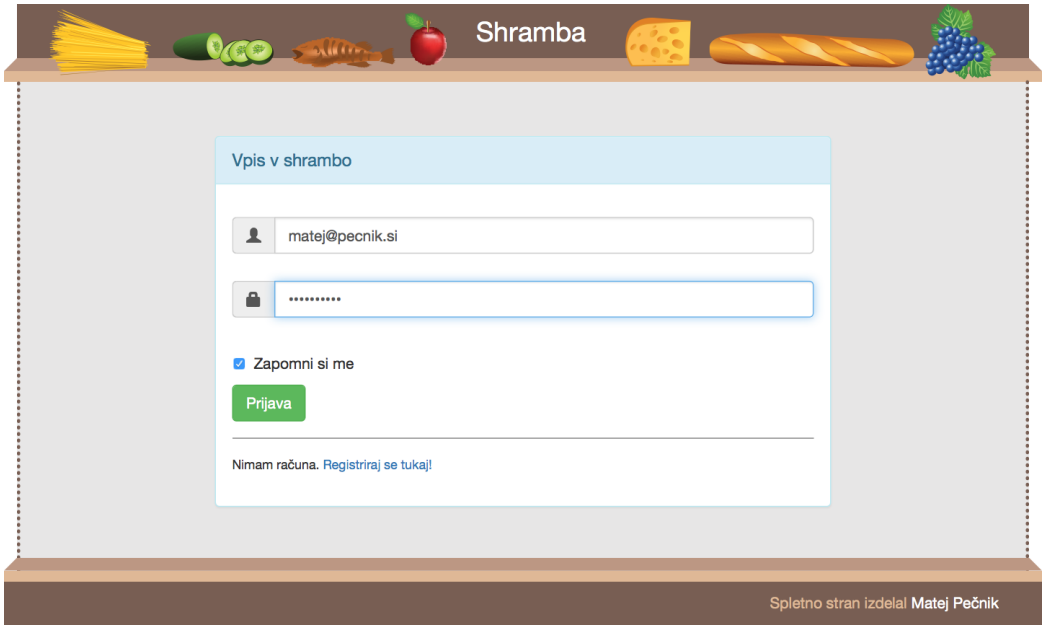
5.1.1 Predstavitev vmesnika za registracijo in prijavo

Pri prvem obisku spletne aplikacije se moramo registrirati. Kot lahko vidimo na Sliki 5.1, pri registraciji vpišemo osebne podatke ter s pritiskom na gumb "Registriraj me" sprožimo zahtevo za registracijo.



Slika 5.1: Spletni vmesnik za registracijo.

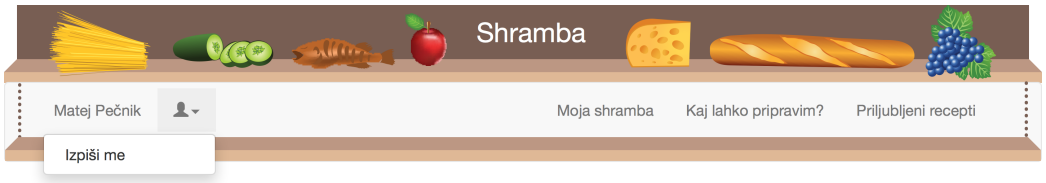
Po postopku registracije sledi postopek prijave. Kot vidimo na Sliki 5.2, se v Shrambo prijavimo s svojim elektronskim naslovom (Email) ter geslom. Pred prijavo lahko označimo še potrditveno polje "Zapomni si me", s katerim dosežemo, da se izpolnjeni podatki pri naslednjem obisku ohranijo. V Shrambo se prijavimo s pritiskom na gumb "Prijava", ki se na sliki nahaja pod potrditvenim poljem. Če si želimo ustvariti nov račun pa to storimo s pritiskom na povezavo "Registriraj se tukaj!", ki uporabnika preusmeri na vmesnik za registracijo.



Slika 5.2: Spletni vmesnik za prijavo.

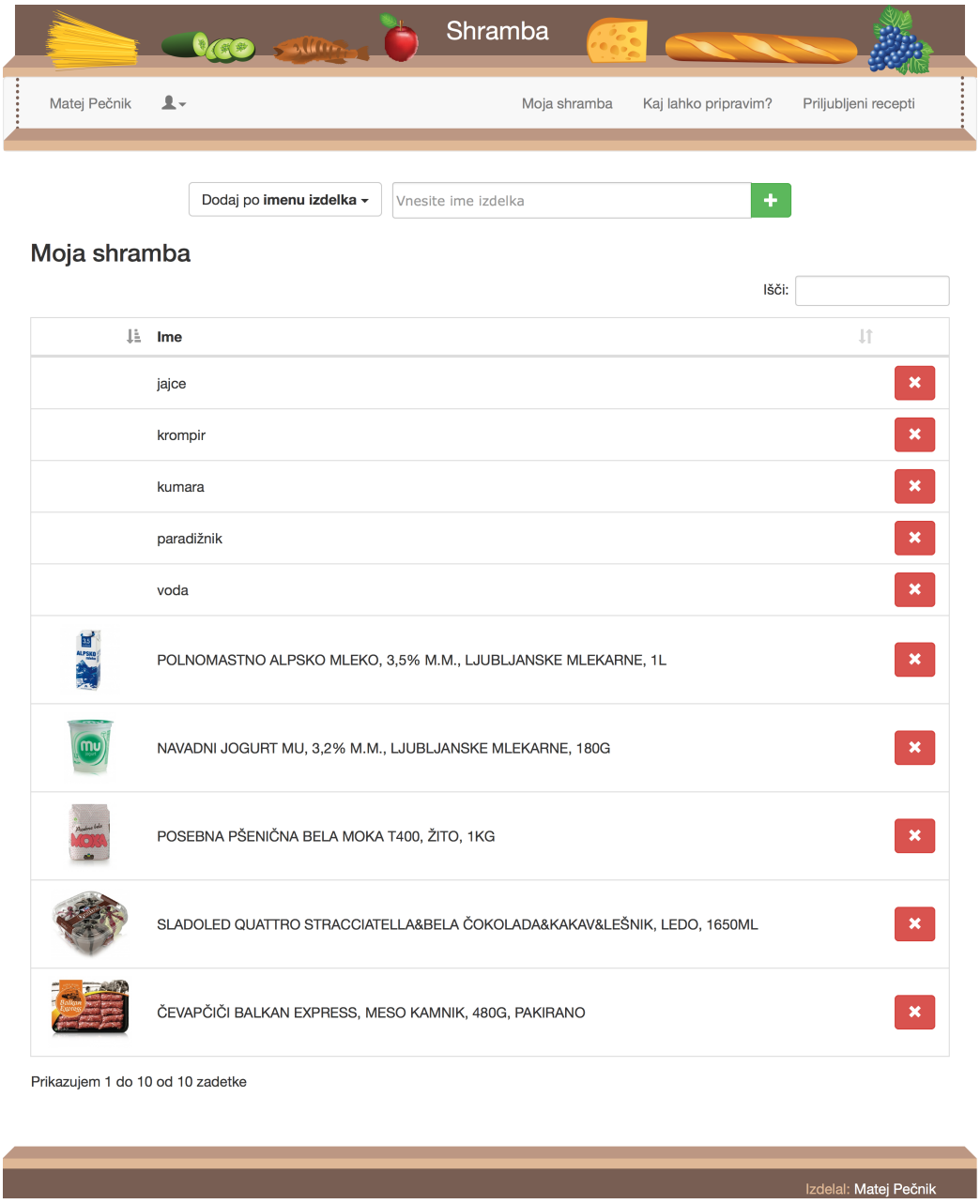
5.1.2 Predstavitev uporabniškega vmesnika

Po uspešni prijavi v sistem se nam prikaže uporabniški vmesnik naše Shrambe. Uporabniški vmesnik je zgrajen iz treh delov (navigacijski meni, glavni del in noga). Slika 5.3 prikazuje navigacijski meni, preko katerega lahko dostopamo do shrambe, ponujenih receptov ter priljubljenih receptov. Poleg tega pa meni vsebuje tudi gumb za odjavo "Izpiši me", ki se nahaja v spustnem meniju, ki je v levem kotu zraven imena in priimka uporabnika.



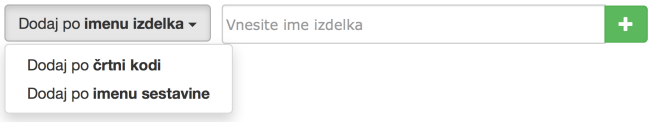
Slika 5.3: Navigacijski meni.

Stran "Moja shramba" prikazuje Slika 5.4. Glavni namen te strani je pregled, dodajanje in odstranjevanje izdelkov ter sestavin, ki jih imamo v shrambi.



Slika 5.4: Stran "Moja shramba".

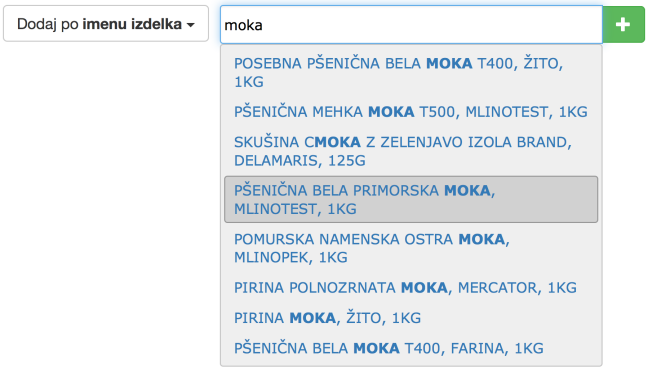
Preko spustnega seznama, ki se nahaja pod navigacijskim menijem, izberemo na kakšen način bi radi izdelek oziroma sestavino dodali. Slika 5.5 prikazuje izbiro načina vnosa izdelkov ter sestavin.



Slika 5.5: Izbira načina vnosa.

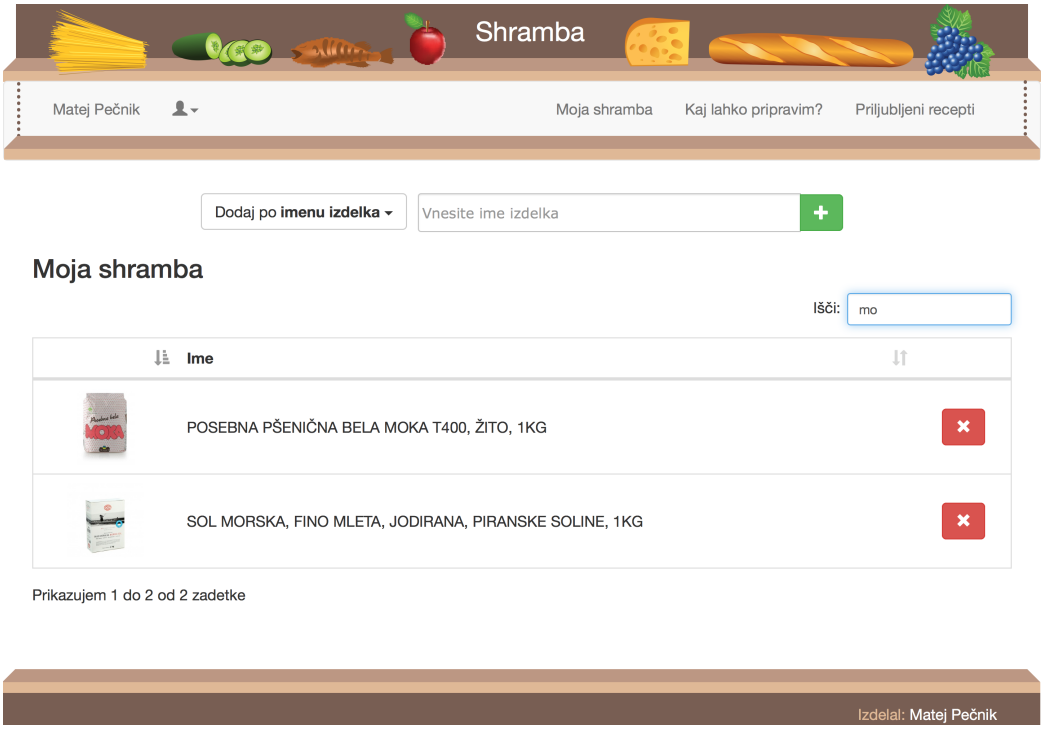
Po izbiri načina vnosa s pomočjo iskalnika, ki se nahaja zraven spustnega seznama, poiščemo željen izdelek oziroma sestavino. Iskalnik nam na podlagi vhoda s pomočjo funkcije za avtomatsko dokončanje (ang. autocomplete) vrne priporočen seznam sestavin oziroma izdelkov, ki vsebujejo iskani niz. Po končanem iskanju izdelek oziroma sestavino dodamo s pritiskom na gumb, ki se nahaja na desni strani iskalnika.

Slika 5.6 prikazuje primer iskanja izdelka, ki vsebuje niz “moka”. Po dodajanju izdelka oziroma sestavine, posamezni dodan vnos najdemo v tabeli pod iskalnikom.



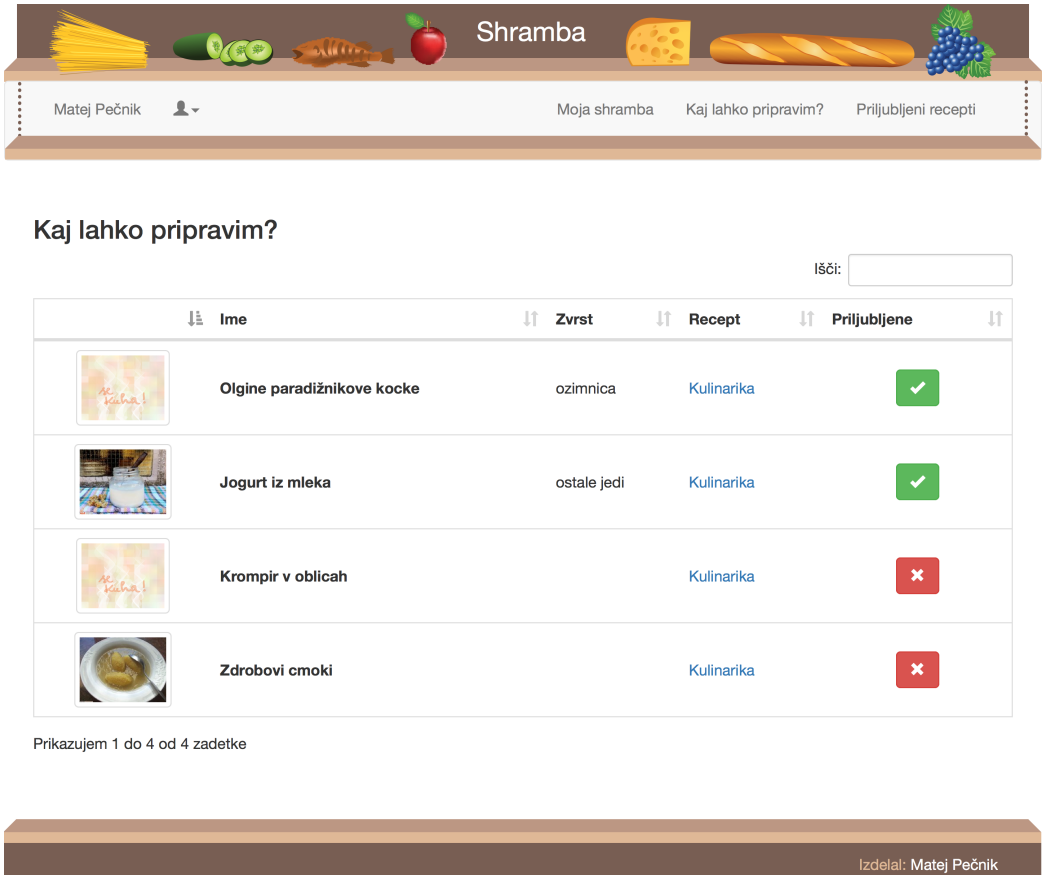
Slika 5.6: Iskalnik izdelkov oziroma sestavin.

Tabela predstavlja našo shrambo. Posamezna sestavina je predstavljena z vrstico, ki vsebuje ime sestavine. Poleg imena pa na desni strani najdemo še gumb, preko katerega lahko odstranimo le-to. Prav tako je predstavljen tudi posamezen izdelek, le da za razliko od sestavine vsebuje še sliko. Ker lahko shramba hitro vsebuje veliko izdelkov ter je tako posledično nepregledna, tabela omogoča urejanje po imenu. Poleg urejanja pa lahko po shrambi tudi iščemo s pomočjo iskalnika, ki ga najdemo na zgornji desni strani nad tabelo. Primer rezultata iskanja prikazuje Slika 5.7.



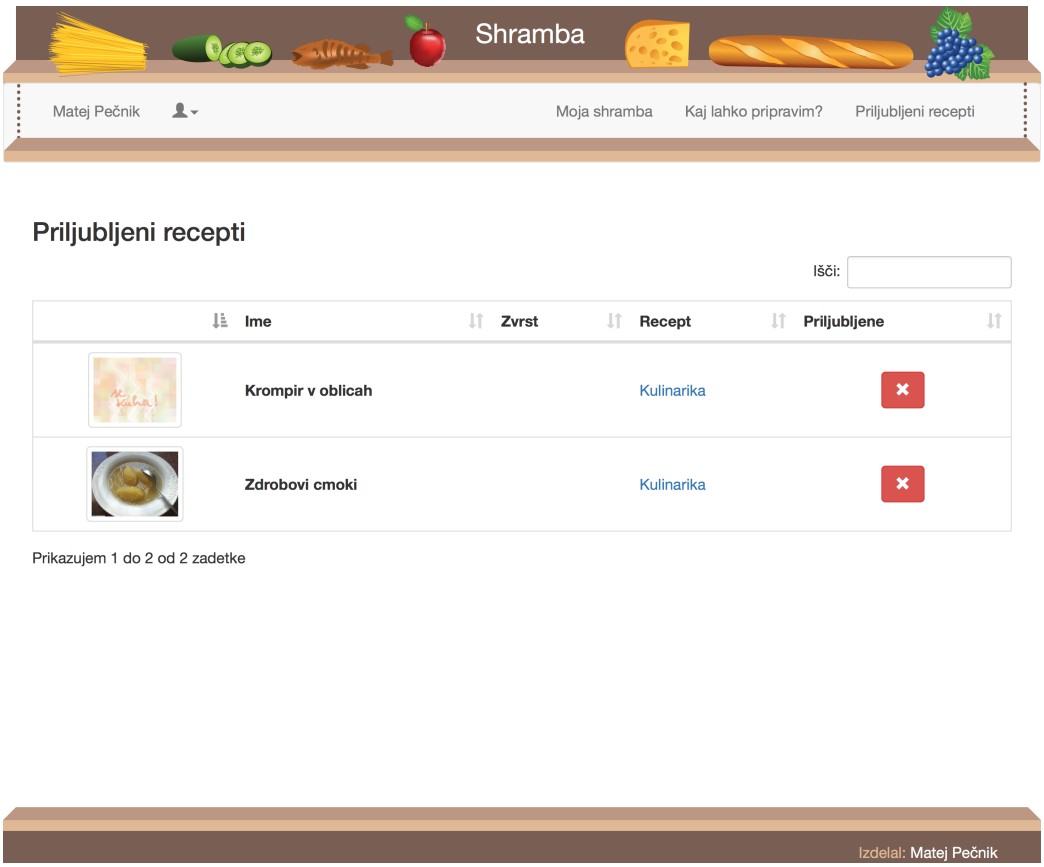
Slika 5.7: Iskalnik po shrambi.

Stran "Kaj lahko pripravim?" prikazuje Slika 5.8. Stran je namenjena prikazu receptov, ki jih lahko naredimo iz sestavin oziroma izdelkov naše shrambe. Recepte, ki jih lahko naredimo, najdemo v tabeli, ki vsebuje pet stolpcev. V prvem stolpcu si lahko ogledamo sliko recepta. Drugi stolpec prikazuje ime recepta, tretji pa zvrst. Pripravo recepta si lahko ogledamo s pritiskom na povezavo recepta, ki jo najdemo v četrtem stolpcu. Zadnji stolpec pa nam omogoča recept shraniti med priljubljene. Vsak stolpec omogoča urejanje, kar nam omogoča preglednejši prikaz receptov. Poleg urejanja se na strani nahaja tudi iskalnik, ki ga najdemo v zgornjem delu strani nad tabelo. Iskalnik omogoča iskanje po množici receptov, kar nam omogoča še hitrejše iskanje jedi, ki bi si jo želeli pripraviti.



Slika 5.8: Stran "Kaj lahko pripravim?".

Priljubljene recepte najdemo na strani "Priljubljeni recepti", ki jo prikazuje Slika 5.9. Na strani si lahko ogledamo recepte, ki smo si jih označili kot priljubljene. Prav tako kot na straneh "Moja shramba" in "Kaj lahko pripravim?" lahko priljubljene recepte uredimo na podlagi stolpca. Poleg urejanja pa imamo tudi možnost iskanja po množici receptov. Priljubljene recepte si lahko tudi ogledamo s pritiskom na povezavo, ki se nahaja zraven zvrsti recepta. Če želimo recept odstraniti iz priljubljenih, to storimo s pritiskom na gumb "X", ki se nahaja na skrajni desni strani recepta.



Slika 5.9: Stran "Priljubljeni recepti".

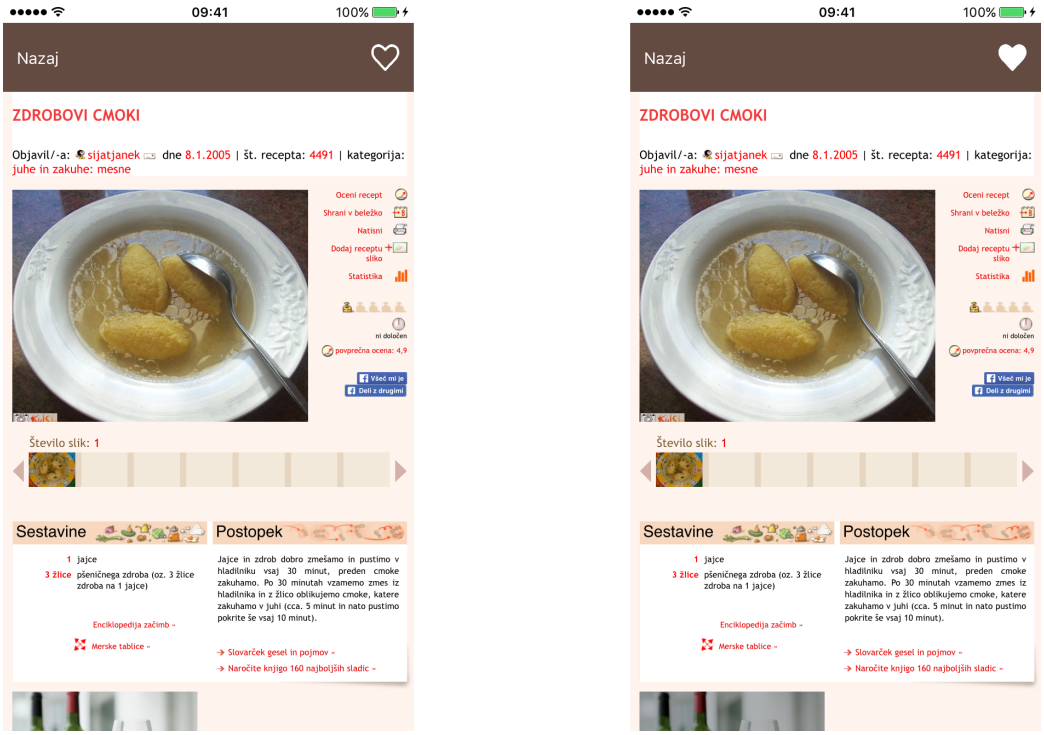
5.2 Mobilna aplikacija

Ob vstopu v aplikacijo se nam prikaže prvi pogled (angl. view). Kot prikazuje Slika 5.10, nam ta prikazuje recepte, ki jih lahko naredimo.

Slika 5.10: Prikaz receptov, ki jih lahko naredimo.

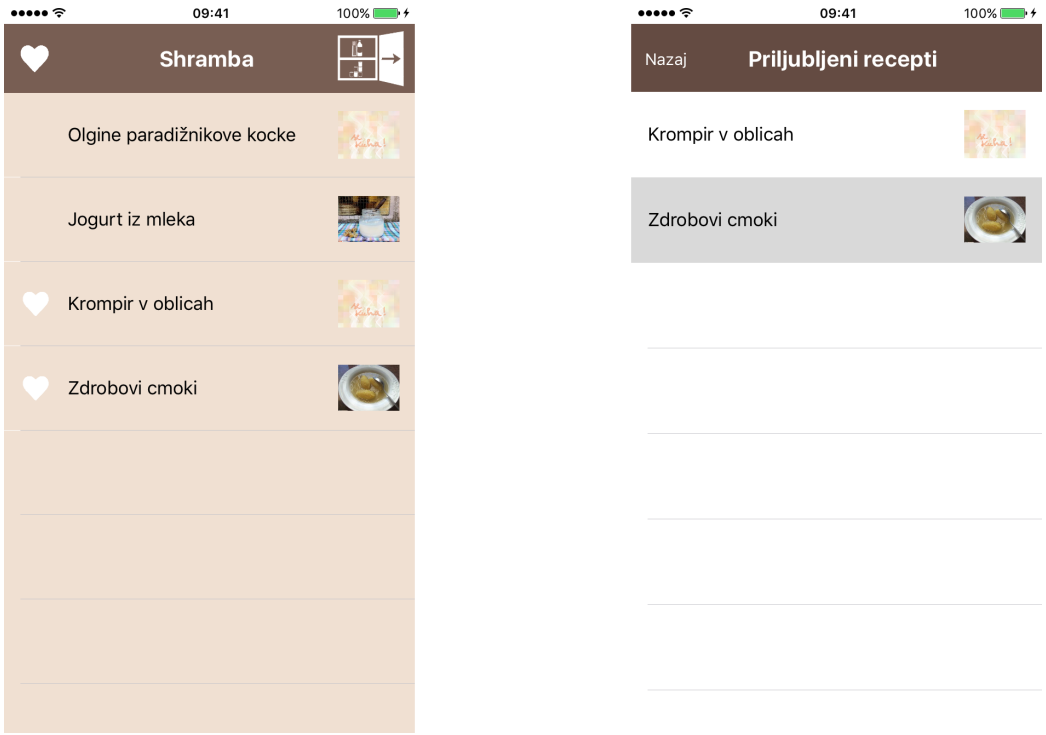
Ob pritisku na vrstico posameznega recepta (desni del Slike 5.10), se nam prikaže pogled za pregled recepta, ki je prikazan na Sliki 5.11. Kot prikazuje Slika 5.11, ta pogled prikazuje podrobne informacije o izbranem receptu. Poleg samega pregleda informacij imamo možnost dodajanja recepta med priljubljene. S pritiskom na gumb z ikono ♡ (prazno srce), ki se nahaja na zgornjem desnem delu pogleda (levi del slike), se nam srce napolni (zgornji desni del desnega dela slike), kar pomeni, da smo recept dodali med priljubljene. Če želimo recept odstraniti iz priljubljenih, to storimo s ponovnim pritiskom na srce. Na prvi pogled receptov se vrnemo s pritiskom na gumb

"Nazaj".



Slika 5.11: Pogled za prikaz podrobnih podatkov o receptu.

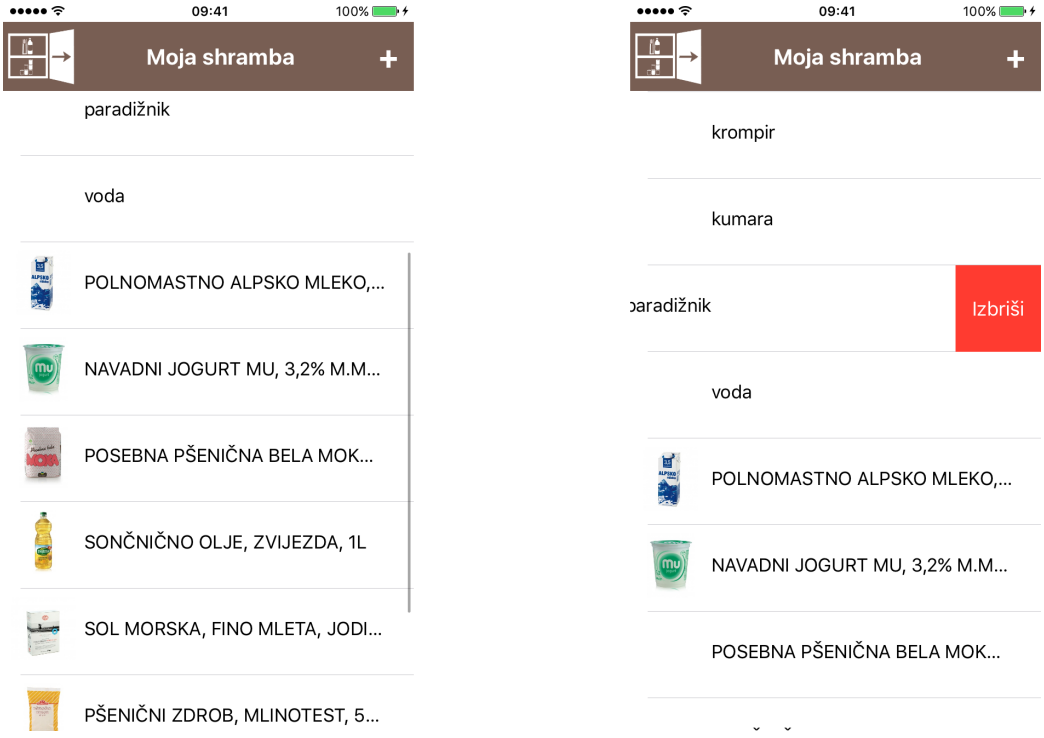
Pri prvem pogledu, ki prikazuje recepte, se priljubljenim receptom (levi del Slike 5.12) zraven imena prikaže ikona ♥ (polno srce). Pri večji količini receptov, ki jih lahko naredimo, postanejo priljubljeni recepti nepregledni, zato si lahko priljubljene recepte ogledamo na pogledu "Priljubljeni recepti". Do pogleda pridemo tako, da pritisnemo na gumb z ikono ♥ (polno srce), ki se nahaja na skrajni zgornji levi strani levega dela Slike 5.12. Pogled "Priljubljeni recepti" se na Sliki 5.12 nahaja na desnem delu. Pri tem pogledu si lahko s pritiskom na recept, le-tega ogledamo ali pa se s pritiskom na gumb "Nazaj" vrnemo nazaj na prvi pogled receptov.



Slika 5.12: Prikaz priljubljenih receptov.

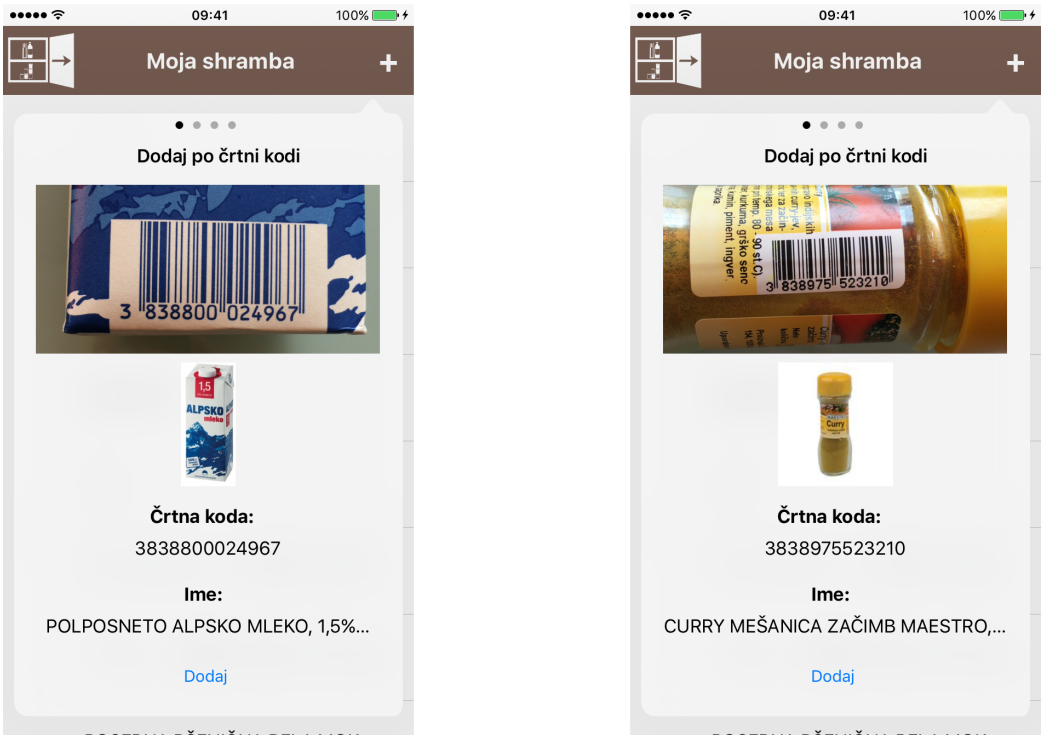
Na pogledu receptov (levi del Slike 5.12) lahko s potegom prsta v levo oziroma s pritiskom na gumb, ki prikazuje shrambo (skrajno desni del levega dela slike), pridemo do pogleda "Moja shramba".

Pogled "Moja shramba" (Slika 5.13) omogoča pregled sestavin in izdelkov, ki jih imamo v shrambi. Kot prikazuje desni del Slike 5.13, lahko posamezno sestavino ali izdelek, ki smo ga porabili, izbrišemo s potegom (angl. swipe) prsta po vrstici izdelka v levo. Prikaže se nam gumb "Izbriši", s katerim ob pritisku nanj potrdimo akcijo brisanja.



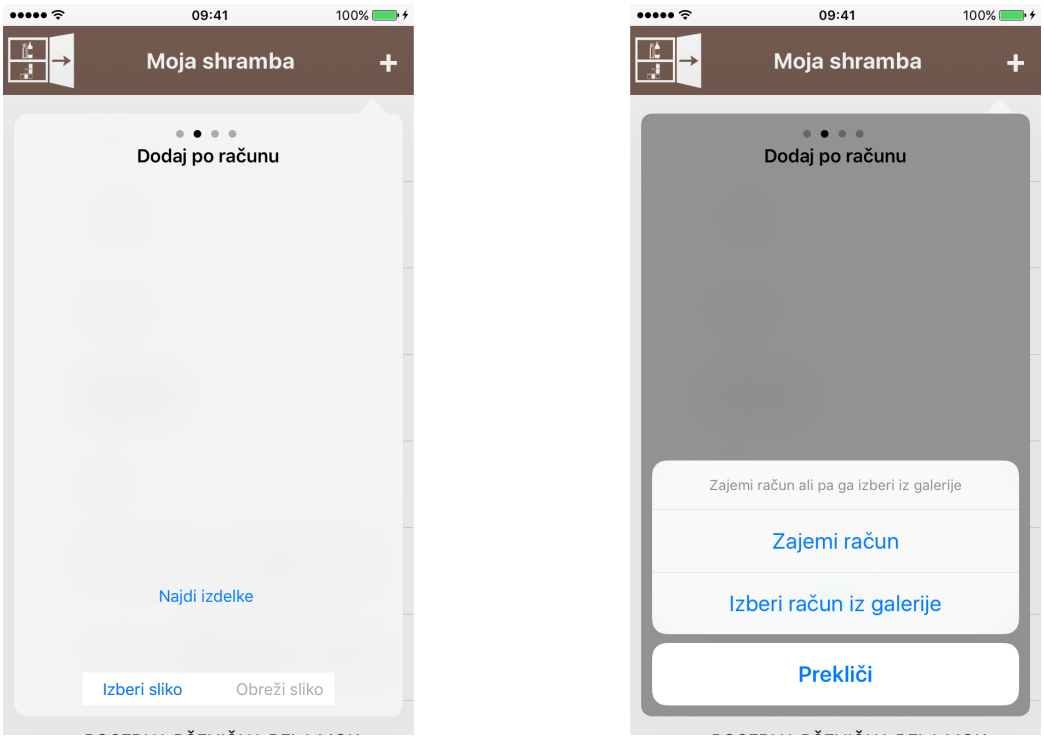
Slika 5.13: Prikaz in izbris izdelkov in sestavin shrambe.

Poleg pregleda ter brisanja izdelkov in sestavin pa lahko le-te tudi dodajamo. To storimo s pritiskom na gumb "+", ki se nahaja na zgornji desni strani pogleda "Moja shramba". Prikaže se nam pojavno okno (angl. popover), ki nam omogoča dodajanje na različne možne načine. Med različnimi načini preklapljamo s pomočjo potega (angl. swipe) prsta v levo oziroma desno. Kot prikazuje Slika 5.14, je prvi od ponujenih načinov dodajanje izdelka po črtni kodi. Pri tem načinu aplikacija uporablja vgrajeno kamero naprave. Mobilno napravo približamo črtni kodi izdelka ter poskrbimo, da se ta nahaja v okvirju aplikacije, ki se nahaja pod napisom "Dodaj po črtni kodi". Po uspešni prepoznavi črtne kode aplikacija rezultat prikaže pod okvirjem. Najden izdelek dodamo s pritiskom na gumb "Dodaj", ki se nahaja pod imenom najdenega izdelka. Slika 5.14 prikazuje primera, pri katerih dodamo izdelek z uporabo skeniranja črtne kode.



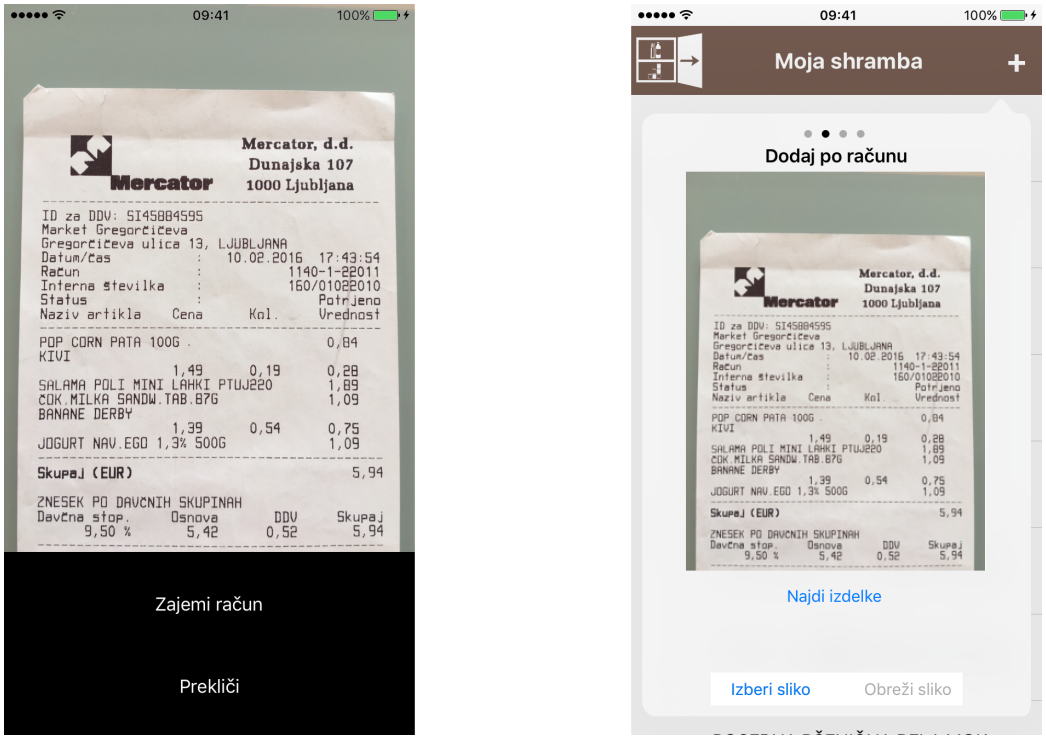
Slika 5.14: Primera dodajanja izdelkov po črtni kodi

Pri drugem načinu dodamo izdelke preko računa, ki ga zajamemo s pomočjo vgrajene kamere naprave. Kot prikazuje Slika 5.15, je prvi korak izbira slike računa. Po pritisku gumba "Izberi sliko" (levi del slike) se nam prikažeta možnosti (desni del slike), preko katere pridobimo sliko računa. Prva možnost nam s pomočjo vgrajene kamere naprave omogoča zajem računa. Druga možnost pa nam omogoča izbiro že zajete slike računa iz galerije slik, ki se nahaja na mobilni napravi.



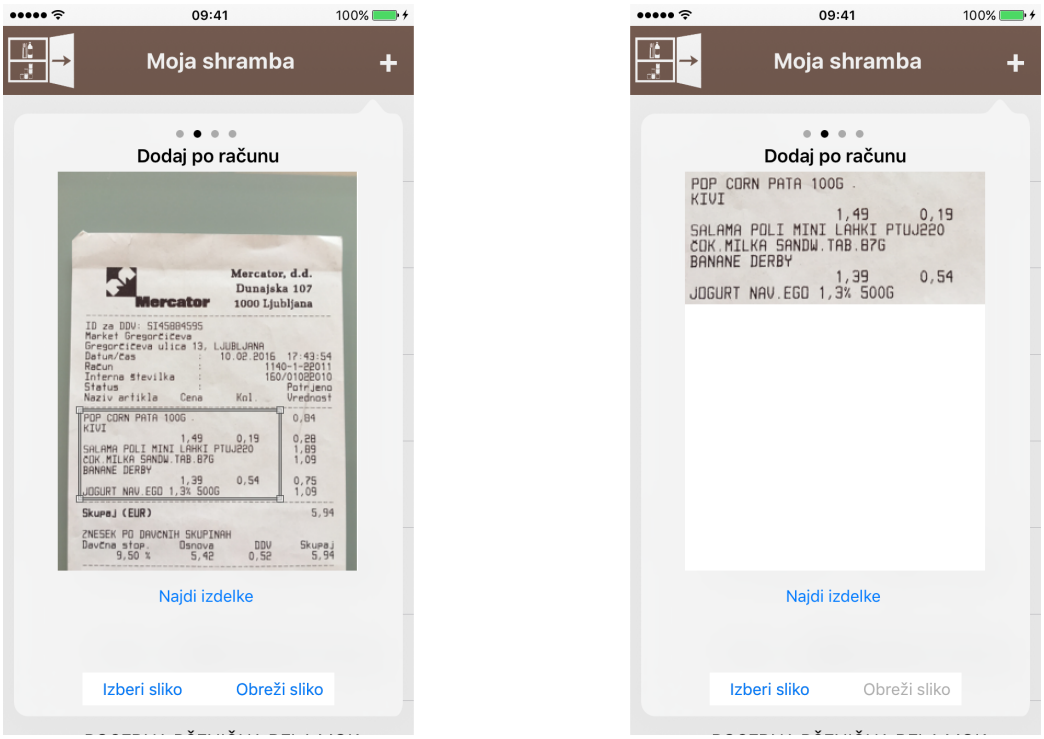
Slika 5.15: Prvi korak dodajanja izdelkov po računu.

Kot prikazuje levi del Slike 5.16, se nam pri izbiri prve možnosti vključi kamera mobilne naprave, preko katere zajamemo sliko. Na voljo imamo možnost "Zajemi račun" ter možnost preklica "Prekliči". Pri izbiri prve možnosti zajamemo sliko računa, ki se nam nato naloži na površino pogleda (desni del slike), kjer je pripravljena za nadaljnjo obdelavo. Izbira druge možnosti pa nas vrne nazaj na korak "Dodaj po računu" (levi del Slike 5.15).



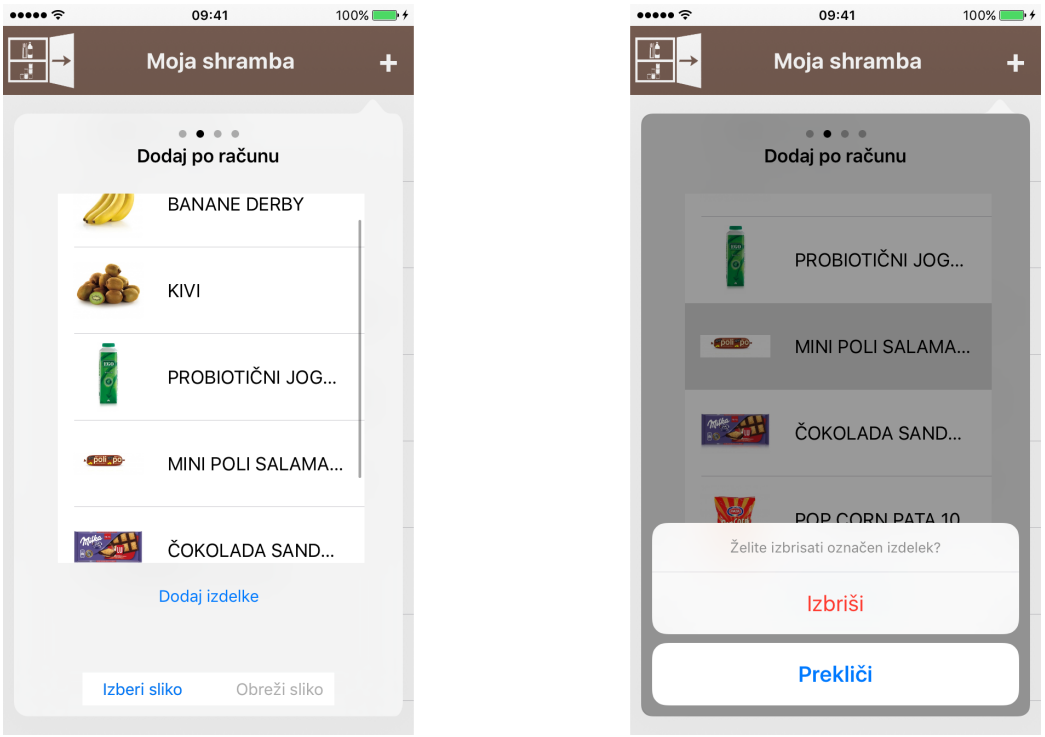
Slika 5.16: Zajem in prikaz slike zajetega računa.

Kot prikazuje Slika 5.17, je v drugem koraku naša naloga, da označimo območje na računu, kjer se nahajajo zapisi izdelkov, ter označeno obrežemo. To storimo s potegom prsta po površini, kjer se nahaja slika. S pomočjo potega prsta se ustvari okvir (levi del slike), ki ga lahko po želji razširimo. Ko smo zadovoljni z označenim, pritisnemo na gumb “Obreži sliko”. Po pritisku se nam prikaže obrezana slika (desni del slike), ki jo lahko po želji z istim postopkom še obrežemo. Ko smo z izbranim zadovoljni, sledi tretji korak, iskanje izdelkov iz označenega. To storimo s pritiskom na gumb “Najdi izdelke”.



Slika 5.17: Prikaz koraka označitve območja računa ter obrezovanja.

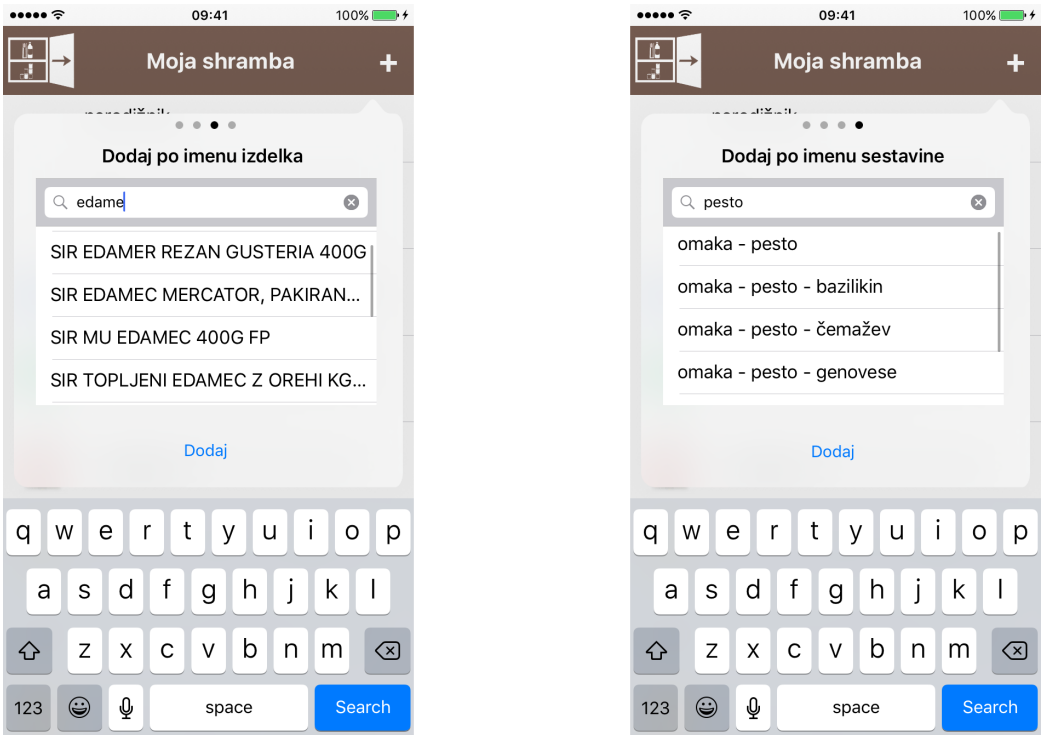
Kot prikazuje Slika 5.18 (levi del slike), je rezultat zgoraj opisane akcije seznam najdenih izdelkov, ki so prikazane v tabeli. V kolikor so vsi izdelki v tabeli pravi, jih dodamo v shrambo. To storimo s pritiskom na gumb "Dodaj izdelke". Če se na seznamu znajde napačen izdelek, ga lahko odstranimo s pritiskom na vrstico izdelka. Po pritisku se nam prikaže opozorilo o akciji (desni del slike). Če želimo izdelek izbrisati, pritisnemo na "Izbriši". Če pa želimo brisanje preklicati, to storimo s pritiskom na "Prekliči".



Slika 5.18: Prikaz in odstranjevanje najdenih izdelkov.

Tretji način dodajanja je dodajanje izdelkov po imenu izdelka. Kot lahko vidimo na desnem delu Slike 5.19, željen izdelek poiščemo s pomočjo iskalnika. Pri vnosu izdelka nam iskalnik sam predlaga možne zadetke, ki vsebujejo iskan niz. Iskan izdelek izberemo s pritiskom na zadetek. V shrambo ga dodamo s pritiskom na gumb "Dodaj".

Četrti način pa nam omogoča dodajanje sestavine. Kot vidimo na levem delu Slike 5.19, to storimo podobno kot pri tretjem načinu. V iskalnik vnesemo ime sestavine, ki jo želimo dodati. Iskalnik pri vnosu predlaga zadetke, ki vsebujejo iskan niz. Želeno sestavino izberemo s pritiskom nanjo. Dodamo pa jo s pritiskom na gumb "Dodaj".



Slika 5.19: Prikaz iskanja izdelkov in sestavin.

Poglavje 6

Zaključek

V tem poglavju pa bomo predstavili sklepne ugotovitve, do katerih smo prišli med razvojem priporočilnega sistema, in ideje za nadaljnje delo.

6.1 Sklepne ugotovitve

V diplomskem delu smo predstavili delovanje ter razvoj avtomatiziranega sistema za priporočanje receptov. Predstavili smo drevo sestavin, ki predstavlja glavno podatkovno strukturo za hranjenje sestavin. Nadaljevali smo s predstavitvijo podatkovne zbirke, ki je sestavljena iz treh skupin. Vozlišče teh skupin predstavlja tabela Ingredient, ki vsebuje drevo sestavin. Drevo sestavin uporabimo pri iskalniku sestavin, ki na podlagi vhoda ugotovi za katero sestavino gre. Ker smo potrebovali podatke o izdelkih in receptih, smo razvili spletna luščilnika. Prvi luščilnik pridobiva podatke o izdelkih iz spletnih virov, medtem ko drugi pridobiva podatke o receptih. Pridobljene podatke nato strukturirano shrani v tabele podatkovne zbirke. Nato je sledila predstavitev iskalnika receptov, katerega glavna naloga je, da iz izdelkov in sestavin, ki jih imamo v shrambi, predlaga recepte, ki jih lahko naredimo. Poleg iskalnika receptov smo predstavili tudi iskalnik izdelkov na računu, katerega naloga je, da poišče izdelek, ki pripada podanemu vhodu. Vhod predstavlja ime izdelka, ki ga dobimo s pomočjo optičnega prepozna-

vanja znakov na računu. Ker rezultat optičnega prepoznavanja znakov ni popolnoma natančen, uporabimo tudi algoritem Levenshteinove razdalje, ki ugotovi podobnost med podanima nizoma. Ker smo želeli do podatkov, ki jih nudi avtomatiziran sistem, dostopati iz več odjemalcev, smo razvili aplikacijski programski vmesnik, ki vsebuje nabor funkcij, s katerimi se izvaja interakcija med odjemalci in strežniškim delom. Predstavili smo tudi razvoj funkcionalnosti spletne aplikacije in mobilne aplikacije, preko katerih pridobimo vpogled v našo shrambo. Na koncu smo predstavili še uporabo obeh uspešno razvitih aplikacij.

6.2 Nadaljnje delo

V nadaljnje bi razvili dodatne module luščilnika receptov, ki bi podatke pridobivali iz več različnih spletnih virov. S tem bi povečali množico receptov, ki jih lahko naredimo. Poleg dodatnih modulov luščilnika receptov bi razvili module luščilnika izdelkov, s katerimi bi povečali množico izdelkov. Večji nabor izdelkov bi omogočal razvoj dodatne funkcionalnosti za primerjavo cen med trgovinami. Izboljšali bi tudi iskalnik receptov. Iskalnik receptov bi priporočal recepte na podlagi določitve nivoja v drevesu sestavin (npr. 1. nivo: (mleko - polnomastno) je enako kot (mleko - posneto), saj sta sestavini na 1. nivoju enaki (mleko)). Optimizirali bi algoritem Levenshteinove razdalje, saj lahko kaj kmalu ugotovimo, da pri hranjenju vmesnih rezultatov ne potrebujemo celotne matrike, zato bi lahko zmanjšali prostorsko zahtevnost algoritma. V aplikacijskem programskem vmesniku bi dodali predpomnenje (angl. caching) ter na ta način povečali odzivnost na zahtevke. Pri optičnem prepoznavanju znakov na mobilni aplikaciji pa bi Tesseractovo slovensko jezikovno datoteko slv.traineddata nadomestili z novo datoteko. Novo jezikovno datoteko bi pridobili z učenjem znakov iz množice slik računa. Pri tem bi pridobili boljše rezultate prepoznavanja izdelkov na računu.

Literatura

[1] Apple Inc. *The Swift Programming Language (Swift 3 beta)*. Apple Inc., 2016.

[2] AV Foundation. [Online]. Dosegljivo: `https://developer.apple.com/av-foundation`. [Dostopano 13. 6. 2016].

[3] AVCaptureDevice. [Online]. Dosegljivo: `https://developer.apple.com/reference/avfoundation/avcapturedevice`. [Dostopano 13. 8. 2016].

[4] AVCaptureSession. [Online]. Dosegljivo: `https://developer.apple.com/reference/avfoundation/avcapturesession`. [Dostopano 13. 8. 2016].

[5] AVCaptureVideoPreviewLayer. [Online]. Dosegljivo: `https://developer.apple.com/reference/avfoundation/avcapturevideopreviewlayer`. [Dostopano 13. 8. 2016].

[6] AVCaptureVideoPreviewLayer. [Online]. Dosegljivo: `https://developer.apple.com/reference/avfoundation/avcapturemetadataoutput`. [Dostopano 13. 8. 2016].

[7] Rakesh Vidya Chandra and Bala Subrahmanyam Varanasi. *Python Requests Essentials*. Packt Publishing - ebooks Account, 2015.

[8] Creating a Barcode and Metadata Reader in iOS. [Online]. Dosegljivo: `https://www.sitepoint.com/creating-barcode-metadata-reader-ios`. [Dostopano 13. 8. 2016].

[9] Janez Demšar. *Python za programerje*. Fakulteta za računalništvo in informatiko, 2009.

[10] Matt Doyle. *Beginning PHP 5.3*. Wrox, 2009.

[11] Paul DuBois. *MySQL (5th Edition) (Developer's Library)*. Addison-Wesley Professional, 2013.

[12] Jon Duckett. *JavaScript and JQuery: Interactive Front-End Web Development*. Wiley, 2014.

[13] Ryan Flores. Getting started with bootstrap 3.3 (code playground book 2), 2014.

[14] Gašper Hafner. *Podobnostne mreže receptov in spletna aplikacija za priporočanje sestavin jedi*. Diplomsko delo, Univerza v Ljubljani, 2011.

[15] Jedel.bi. [Online]. Dosegljivo: `http://www.jedel.bi`. [Dostopano 9. 7. 2016].

[16] Kulinarika.net. [Online]. Dosegljivo: `http://www.kulinarika.net`. [Dostopano 9. 7. 2016].

[17] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[18] Mark Masse. *REST API Design Rulebook*. O'Reilly Media, 2011.

[19] Julie C. Meloni and Michael Morrison. *Sams Teach Yourself HTML and CSS in 24 Hours (Includes New HTML 5 Coverage) (8th Edition)*. Sams Publishing, 2009.

[20] Mercator Spletna trgovina. [Online]. Dosegljivo: `https://trgovina.mercator.si`. [Dostopano 30. 7. 2016].

[21] F.P. Miller, A.F. Vandome, and J. McBrewster. *Levenshtein Distance*. VDM Publishing, 2009.

[22] Mizicapogrnise.si. [Online]. Dosegljivo: `http://www.mizicapogrnise.si`. [Dostopano 9. 7. 2016].

[23] Mojirecepti.com. [Online]. Dosegljivo: `http://www.mojirecepti.com`. [Dostopano 9. 7. 2016].

[24] MySQL Workbench. [Online]. Dosegljivo: `https://www.mysql.com/products/workbench`. [Dostopano 4. 6. 2016].

[25] Vineeth G. Nair. *Getting Started with Beautiful Soup*. Packt Publishing, 2014.

[26] Brett Ohland and Jayant Varma. *Xcode 7 Essentials - Second Edition*. Packt Publishing, 2016.

[27] Okusno.je. [Online]. Dosegljivo: `http://www.okusno.je`. [Dostopano 9. 7. 2016].

[28] Python 3 - MySQL Database Access. [Online]. Dosegljivo: `http://www.tutorialspoint.com/python3/python_database_access.htm`. [Dostopano 11. 6. 2016].

[29] Ray Smith. An overview of the tesseract ocr engine. In *Proc. Ninth Int. Conference on Document Analysis and Recognition (ICDAR)*, pages 629–633, 2007.

[30] Stanford University. Minimum Edit Distance [Online]. Dosegljivo: `https://web.stanford.edu/class/cs124/lec/med.pdf`. [Dostopano 9. 8. 2016].

[31] Swift and Objective-C in the Same Project. [Online]. Dosegljivo: `https://developer.apple.com/library/ios/documentation/`

Swift/Conceptual/BuildingCocoaApps/MixandMatch.html. [Dostopano 13. 8. 2016].

[32] Swizec Teller. *Data Visualization with d3.js*. Packt Publishing, 2013.

[33] Tesseract OCR Tutorial. [Online]. Dosegljivo: https://www.raywenderlich.com/93276/implementing-tesseract-ocr-ios. [Dostopano 14. 8. 2016].

[34] Tuš Cash & Carry. [Online]. Dosegljivo: http://www.tuscc.si. [Dostopano 30. 7. 2016].

[35] W3C. [Online]. Dosegljivo: http://www.w3.org/Addressing. [Dostopano 4. 8. 2016].